Canonical

LXD

Contents

LXD (`[l□ks'di:]`□) is a modern, secure and powerful system container and virtual machine manager.

It provides a unified experience for running and managing full Linux systems inside containers or virtual machines. LXD supports images for a large number of Linux distributions (official Ubuntu images and images provided by the community) and is built around a very powerful, yet pretty simple, REST API. LXD scales from one instance on a single machine to a cluster in a full data center rack, making it suitable for running workloads both for development and in production.

LXD allows you to easily set up a system that feels like a small private cloud. You can run any type of workload in an efficient way while keeping your resources optimized.

You should consider using LXD if you want to containerize different environments or run virtual machines, or in general run and manage your infrastructure in a cost-effective way.

In this documentation

Project and community

LXD is free software and released under AGPL-3.0-only[14] (it may contain some contributions that are licensed under the Apache-2.0 license, see *License and copyright* (page 335)). It's an open source project that warmly welcomes community projects, contributions, suggestions, fixes and constructive feedback.

The LXD project is sponsored by Canonical Ltd[15].

- Ubuntu Code of Conduct[16]
- *Contribute to the project* (page 335)
- Release announcements[17]
- Release tarballs[18]
- *Get support* (page 334)
- Watch tutorials and announcements on YouTube[19]
- Discuss on IRC[20] (see Getting started with IRC[21] if needed)
- Ask and answer questions on the forum[22]

[14] https://www.gnu.org/licenses/agpl-3.0.en.html
[15] https://canonical.com/
[16] https://ubuntu.com/community/ethos/code-of-conduct
[17] https://discourse.ubuntu.com/c/lxd/news/143
[18] https://github.com/canonical/lxd/releases/
[19] https://www.youtube.com/c/LXDvideos
[20] https://web.libera.chat/#lxd
[21] https://discourse.ubuntu.com/t/getting-started-with-irc/37907
[22] https://discourse.ubuntu.com/c/lxd/126

1. Tutorials

The following tutorial guides you through installing and initializing LXD, creating and configuring some instances, interacting with the instances, and creating snapshots:

1.1. First steps with LXD

This tutorial guides you through the first steps with LXD. It covers installing and initializing LXD, creating and configuring some instances, interacting with the instances, and creating snapshots.

After going through these steps, you will have a general idea of how to use LXD, and you can start exploring more advanced use cases!

> **Note**
>
> Ensure that you have 20 GiB free disk space before starting this tutorial.

1.1.1. Install and initialize LXD

The easiest way to install LXD is to install the snap package. If you prefer a different installation method, or use a Linux distribution that is not supported by the snap package, see *How to install LXD* (page 28).

1. Install `snapd`:

   1. Run `snap version` to find out if snap is installed on your system:

      ```
      ~$ snap version snap 2.63+24.04ubuntu0.1snapd
      2.63+24.04ubuntu0.1series 16ubuntu 24.04kernel 5.15.0-117-generic
      ```

      If you see a table of version numbers, snap is installed and you can continue with the next step of installing LXD.

   2. If the command returns an error, run the following commands to install the latest version of `snapd` on Ubuntu:

      ```
      sudo apt update
      sudo apt install snapd
      ```

      > **Note**
      >
      > For other Linux distributions, see the installation instructions[23] in the Snapcraft documentation.

2. Enter the following command to install LXD:

   ```
   sudo snap install lxd
   ```

[23] https://snapcraft.io/docs/installing-snapd

If you get an error message that the LXD snap is already installed, run the following command to refresh it and ensure that you are running an up-to-date version:

```
sudo snap refresh lxd
```

3. Check if the current user is part of the `lxd` group (the group was automatically created during the previous step):

```
getent group lxd | grep "$USER"
```

If this command returns a result, you're set up correctly and can continue with the next step.

If there is no result, enter the following commands to add the current user to the `lxd` group (which is needed to grant the user permission to interact with LXD):

```
sudo usermod -aG lxd "$USER"
newgrp lxd
```

4. Enter the following command to initialize LXD:

```
lxd init --minimal
```

This will create a minimal setup with default options. If you want to tune the initialization options, see *How to initialize LXD* (page 35) for more information.

5. LXD supports both *Containers and VMs* (page 346). For LXD virtual machines, your host system must be capable of KVM virtualization. To check this, run the following command:

```
lxc info | grep -FA2 'instance_types'
```

The following output indicates that your host system is capable of virtualization:

```
instance_types:
- container
- virtual-machine
```

If `virtual-machine` fails to appear in the output, this indicates that the host system is not capable of virtualization. In this case, LXD can only be used for containers. You can proceed with this tutorial to learn about using containers, but disregard the steps that refer to virtual machines.

1.1.2. Launch and inspect instances

LXD is image based and can load images from different image servers. In this tutorial, we will use the official `ubuntu:`[24] image server.

You can list all images (long list) that are available on this image server with:

```
lxc image list ubuntu:
```

You can list the images used in this tutorial with:

[24] https://cloud-images.ubuntu.com/releases/

```
lxc image list ubuntu: 24.04 architecture=$(uname -m)
```

See *Images* (page 148) for more information about the images that LXD uses.

Now, let's start by launching a few instances. With *instance*, we mean either a container or a virtual machine. See *Containers and VMs* (page 346) for information about the difference between the two instance types.

For managing instances, we use the LXD command line client `lxc`. See *About `lxd` and `lxc`* (page 345) if you are confused about when to use the `lxc` command and when to use the `lxd` command.

1. Launch a container called `first` using the Ubuntu 24.04 LTS image:

   ```
   lxc launch ubuntu:24.04 first
   ```

   > **Note**
   >
   > Launching this container takes a few seconds, because the image must be downloaded and unpacked first.

2. Launch a container called `second` using the same image:

   ```
   lxc launch ubuntu:24.04 second
   ```

   > **Note**
   >
   > Launching this container is quicker than launching the first, because the image is already available locally.

3. Copy the first container into a container called `third`:

   ```
   lxc copy first third
   ```

4. Launch a VM called `ubuntu-vm` using the Ubuntu 24.04 LTS image:

   ```
   lxc launch ubuntu:24.04 ubuntu-vm --vm
   ```

   > **Note**
   >
   > Even though you are using the same image name to launch the instance, LXD downloads a slightly different image that is compatible with VMs.

5. Check the list of instances that you launched:

   ```
   lxc list
   ```

   You will see that all but the third container are running. This is because you created the third container by copying the first, but you didn't start it.

   You can start the third container with:

```
lxc start third
```

6. Query more information about each instance with:

```
lxc info first
lxc info second
lxc info third
lxc info ubuntu-vm
```

7. We don't need all of these instances for the remainder of the tutorial, so let's clean some of them up:

   1. Stop the second container:

   ```
   lxc stop second
   ```

   2. Delete the second container:

   ```
   lxc delete second
   ```

   3. Delete the third container:

   ```
   lxc delete third
   ```

   Since this container is running, you get an error message that you must stop it first. Alternatively, you can force-delete it:

   ```
   lxc delete third --force
   ```

See *How to create instances* (page 73) and *How to manage instances* (page 92) for more information.

1.1.3. Configure instances

There are several limits and configuration options that you can set for your instances. See *Instance options* (page 415) for an overview.

Let's create another container with some resource limits:

1. Launch a container and limit it to one vCPU and 192 MiB of RAM:

```
lxc launch ubuntu:24.04 limited --config limits.cpu=1 --config limits.
memory=192MiB
```

2. Check the current configuration and compare it to the configuration of the first (unlimited) container:

```
lxc config show limited
lxc config show first
```

3. Check the amount of free and used memory on the parent system and on the two containers:

```
free -m
lxc exec first -- free -m
lxc exec limited -- free -m
```

> **Note**
>
> The total amount of memory is identical for the parent system and the first container, because by default, the container inherits the resources from its parent environment. The limited container, on the other hand, has only 192 MiB available.

4. Check the number of CPUs available on the parent system and on the two containers:

```
nproc
lxc exec first -- nproc
lxc exec limited -- nproc
```

> **Note**
>
> Again, the number is identical for the parent system and the first container, but reduced for the limited container.

5. You can also update the configuration while your container is running:

   1. Configure a memory limit for your container:

      ```
      lxc config set limited limits.memory=128MiB
      ```

   2. Check that the configuration has been applied:

      ```
      lxc config show limited
      ```

   3. Check the amount of memory that is available to the container:

      ```
      lxc exec limited -- free -m
      ```

      Note that the number has changed.

6. Depending on the instance type and the storage drivers that you use, there are more configuration options that you can specify. For example, you can configure the size (quota) of the root disk device for a VM:

   1. Check the current size of the root disk device of the Ubuntu VM:

      ```
      ~$ lxc exec ubuntu-vm -- df -h Filesystem Size Used Avail Use%
      Mounted on/dev/root 9.6G 1.4G 8.2G 15% /tmpfs 483M 0 483M 0%
      /dev/shmtmpfs 193M 604K 193M 1% /runtmpfs 5.0M 0 5.0M 0%
      /run/locktmpfs 50M 14M 37M 27% /run/lxd_agent/dev/sda15 105M 6.1M
      99M 6% /boot/efi
      ```

   2. Override the size of the root disk device:

      ```
      lxc config device override ubuntu-vm root size=30GiB
      ```

   3. Restart the VM:

```
lxc restart ubuntu-vm
```

4. Check the size of the root disk device again:

```
~$ lxc exec ubuntu-vm -- df -h Filesystem Size Used Avail Use%
Mounted on/dev/root 29G 1.4G 28G 5% /tmpfs 483M 0 483M 0%
/dev/shmtmpfs 193M 588K 193M 1% /runtmpfs 5.0M 0 5.0M 0%
/run/locktmpfs 50M 14M 37M 27% /run/lxd_agent/dev/sda15 105M 6.1M
99M 6% /boot/efi
```

See *How to configure instances* (page 84) and *Instance configuration* (page 414) for more information.

1.1.4. Interact with instances

You can interact with your instances by running commands in them (including an interactive shell) or accessing the files in the instance.

Start by launching an interactive shell in your instance:

1. Run the bash command in your container:

```
lxc exec first -- bash
```

2. Enter some commands, for example, display information about the operating system:

```
cat /etc/*release
```

3. Exit the interactive shell:

```
exit
```

Instead of logging on to the instance and running commands there, you can run commands directly from the host.

For example, you can install a command line tool on the instance and run it:

```
lxc exec first -- apt-get update
lxc exec first -- apt-get install sl -y
lxc exec first -- /usr/games/sl
```

See *How to run commands in an instance* (page 111) for more information.

You can also access the files from your instance and interact with them:

1. Pull a file from the container:

```
lxc file pull first/etc/hosts .
```

2. Add an entry to the file:

```
echo "1.2.3.4 my-example" >> hosts
```

3. Push the file back to the container:

```
lxc file push hosts first/etc/hosts
```

4. Use the same mechanism to access log files:

```
lxc file pull first/var/log/syslog - | less
```

> **Note**
>
> Press `q` to exit the `less` command.

See *How to access files in an instance* (page 105) for more information.

1.1.5. Manage snapshots

You can create a snapshot of your instance, which makes it easy to restore the instance to a previous state.

1. Create a snapshot called "clean":

```
lxc snapshot first clean
```

2. Confirm that the snapshot has been created:

```
lxc list first
lxc info first
```

> **Note**
>
> `lxc list` shows the number of snapshots. `lxc info` displays information about each snapshot.

3. Break the container:

```
lxc exec first -- rm /usr/bin/bash
```

4. Confirm the breakage:

```
lxc exec first -- bash
```

> **Note**
>
> You do not get a shell, because you deleted the `bash` command.

5. Restore the container to the state of the snapshot:

```
lxc restore first clean
```

6. Confirm that everything is back to normal:

```
lxc exec first -- bash
exit
```

7. Delete the snapshot:

```
lxc delete first/clean
```

See *Use snapshots for instance backup* (page 128) for more information.

1.1.6. Next steps

Now that you've done your first experiments with LXD, you should read up on important concepts in the *Explanation* (page 345) section and check out the *How-to guides* (page 28) to start working with LXD!

1.2. Getting started with the UI

This tutorial gives a quick introduction to using the LXD UI. It covers installing and initializing LXD, getting access to the UI, and carrying out some standard operations like creating, configuring, and interacting with instances, configuring storage, and using projects.

After going through these steps, you will have a general idea of how to use LXD through its UI, and you can start exploring more advanced use cases!

> **Note**
>
> Ensure that you have 20 GiB free disk space before starting this tutorial.

1.2.1. Install and initialize LXD

The easiest way to install LXD is to install the snap package. If you prefer a different installation method, or use a Linux distribution that is not supported by the snap package, see *How to install LXD* (page 28).

1. Install `snapd`:

   1. Run `snap version` to find out if snap is installed on your system:

      ```
      ~$ snap version snap 2.63+24.04ubuntu0.1snapd
      2.63+24.04ubuntu0.1series 16ubuntu 24.04kernel 5.15.0-117-generic
      ```

      If you see a table of version numbers, snap is installed and you can continue with the next step of installing LXD.

   2. If the command returns an error, run the following commands to install the latest version of `snapd` on Ubuntu:

      ```
      sudo apt update
      sudo apt install snapd
      ```

      > **Note**
      >
      > For other Linux distributions, see the installation instructions[25] in the Snapcraft documentation.

2. Enter the following command to install LXD:

[25] https://snapcraft.io/docs/installing-snapd

```
sudo snap install lxd
```

If you get an error message that the LXD snap is already installed, run the following command to refresh it and ensure that you are running an up-to-date version:

```
sudo snap refresh lxd
```

3. Check if the current user is part of the `lxd` group (the group was automatically created during the previous step):

```
getent group lxd | grep "$USER"
```

If this command returns a result, you're set up correctly and can continue with the next step.

If there is no result, enter the following commands to add the current user to the `lxd` group (which is needed to grant the user permission to interact with LXD):

```
sudo usermod -aG lxd "$USER"
newgrp lxd
```

4. Enter the following command to initialize LXD:

```
lxd init --minimal
```

This will create a minimal setup with default options. If you want to tune the initialization options, see *How to initialize LXD* (page 35) for more information.

5. LXD supports both *Containers and VMs* (page 346). For LXD virtual machines, your host system must be capable of KVM virtualization. To check this, run the following command:

```
lxc info | grep -FA2 'instance_types'
```

The following output indicates that your host system is capable of virtualization:

```
instance_types:
- container
- virtual-machine
```

If `virtual-machine` fails to appear in the output, this indicates that the host system is not capable of virtualization. In this case, LXD can only be used for containers. You can proceed with this tutorial to learn about using containers, but disregard the steps that refer to virtual machines.

1.2.2. Access the UI

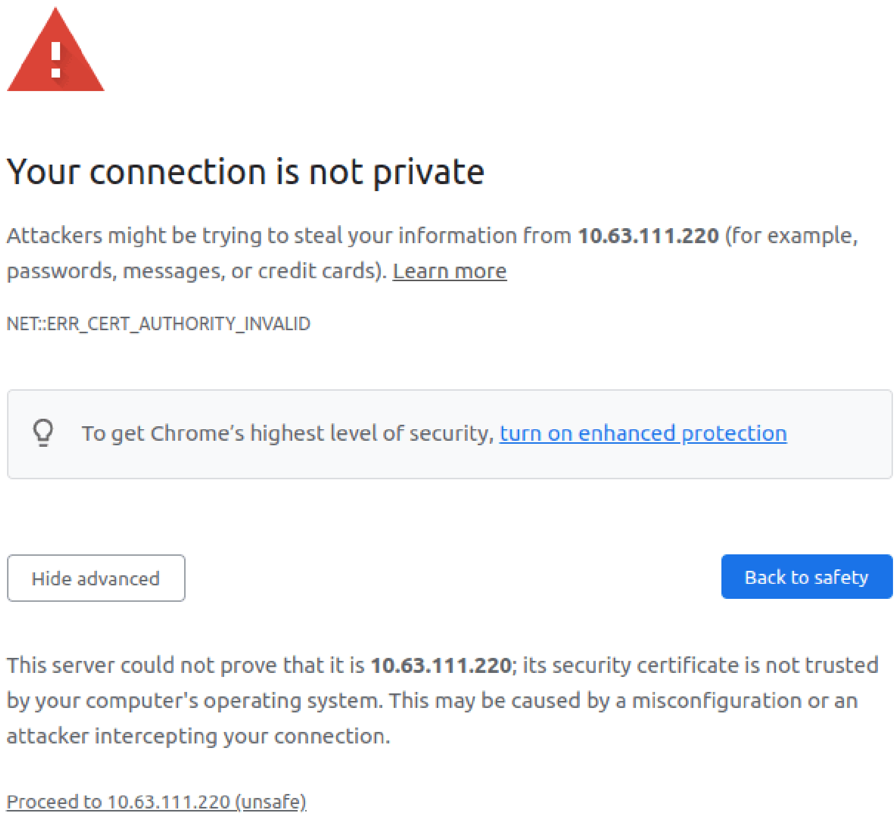
You access the LXD UI through your browser. See *How to access the LXD web UI* (page 40) for more information.

1. Expose LXD to the network by setting the *`core.https_address`* (page 402) server configuration option:

```
lxc config set core.https_address :8443
```

2. Access the UI in your browser by entering the server address (for example, `https://127.0.0.1:8443` for a local server, or an address like `https://192.0.2.10:8443` for a server running on `192.0.2.10`).

   If you have not set up a secure *TLS server certificate* (page 361), LXD uses a self-signed certificate, which will cause a security warning in your browser. Use your browser's mechanism to continue despite the security warning.



3. Set up the certificates that are required for the UI client to authenticate with the LXD server by following the steps presented in the UI.

   You have two options, depending on whether you already have a client certificate selected in your browser:

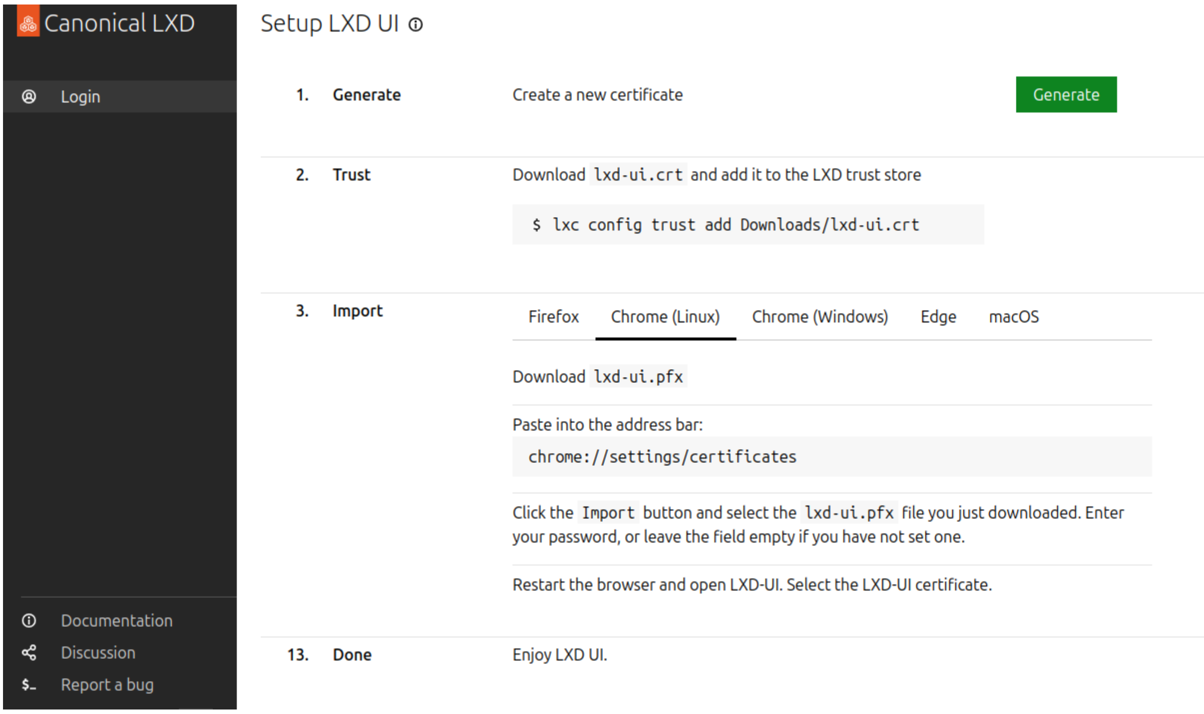
   - If you don't have a certificate yet, click *Create a new certificate* to get instructions for creating a set of certificates, adding the public key to the server's trust store, and adding the private key to your browser.
   - If you already have a client certificate in your browser, select "use an existing certificate" to authorize the certificate with the server and re-use it.

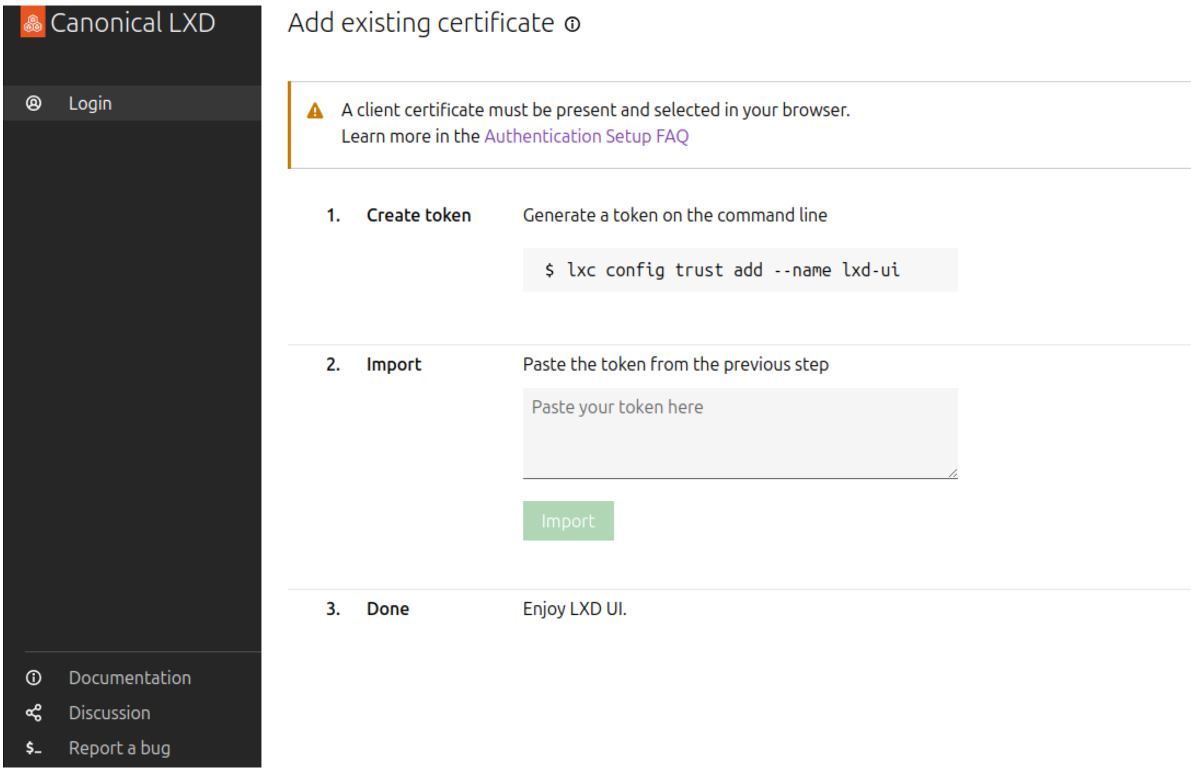
   See *Remote API authentication* (page 358) for more information.

1.2.3. Create and start instances

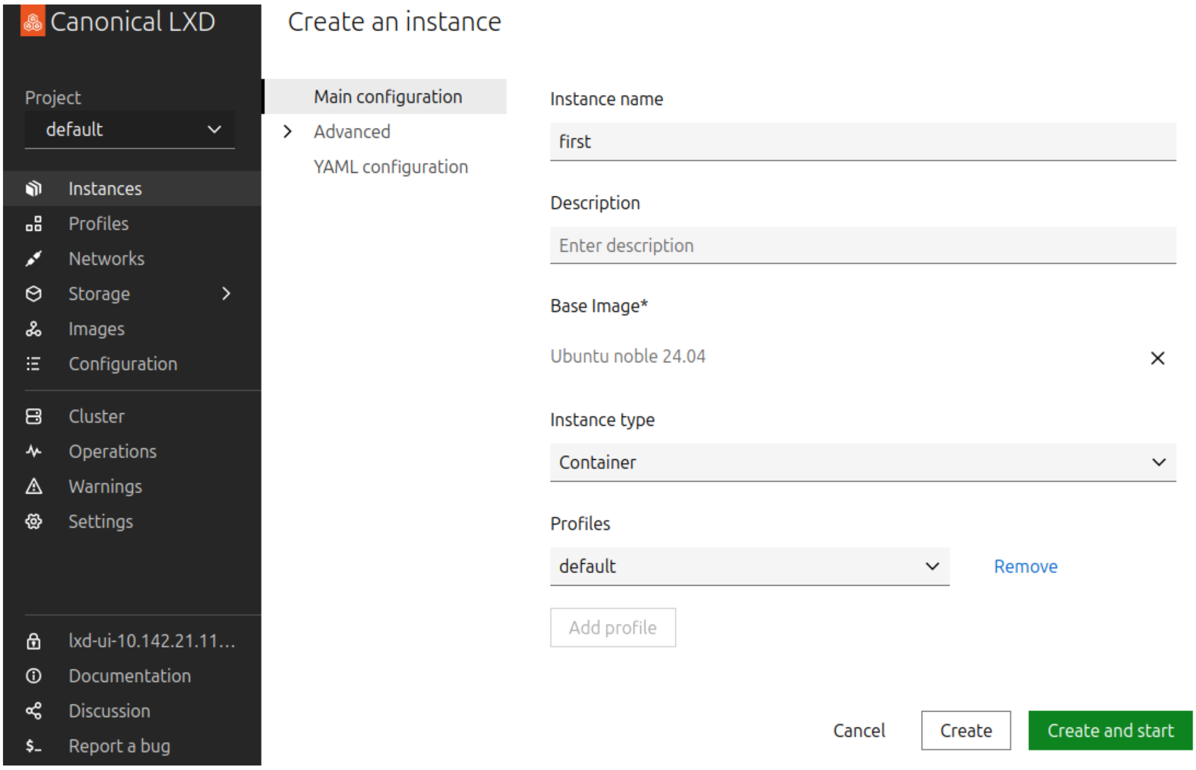
Let's start by launching a few instances. With *instance*, we mean either a container or a virtual machine. See *Containers and VMs* (page 346) for information about the difference between the two instance types.

1. Select *Instances* from the navigation.


Canonical LXD
Login
Documentation
Discussion
Report a bug
Setup LXD UI
1. Generate
Create a new certificate
Generate
2. Trust
Download lxd-ui.crt and add it to the LXD trust store
$ lxc config trust add Downloads/lxd-ui.crt
3. Import
Firefox
Chrome (Linux)
Chrome (Windows)
Edge
macOS
Download lxd-ui.pfx
Paste into the address bar:
chrome://settings/certificates
Click the Import button and select the lxd-ui.pfx file you just downloaded. Enter your password, or leave the field empty if you have not set one.
Restart the browser and open LXD-UI. Select the LXD-UI certificate.
13. Done
Enjoy LXD UI.


Canonical LXD
Login
Documentation
Discussion
Report a bug
Add existing certificate
A client certificate must be present and selected in your browser.
Learn more in the Authentication Setup FAQ
1. Create token
Generate a token on the command line
$ lxc config trust add --name lxd-ui
2. Import
Paste the token from the previous step
Paste your token here
Import
3. Done
Enjoy LXD UI.

2. Click *Create instance* to launch a container called `first` using the Ubuntu 24.04 LTS image:



To select the base image, click *Browse images*. Click *Select* next to the Ubuntu 24.04 LTS image.

> **Note**
>
> The images that are displayed are hosted on pre-configured *Remote image servers* (page 391). You can filter which images are displayed.
>
> You can also upload a custom ISO file to boot from. See *Create a VM that boots from an ISO* (page 80) for more information.

3. To launch the container, click *Create and start*.

> **Note**
>
> Launching this container takes a few seconds, because the image must be downloaded and unpacked first.

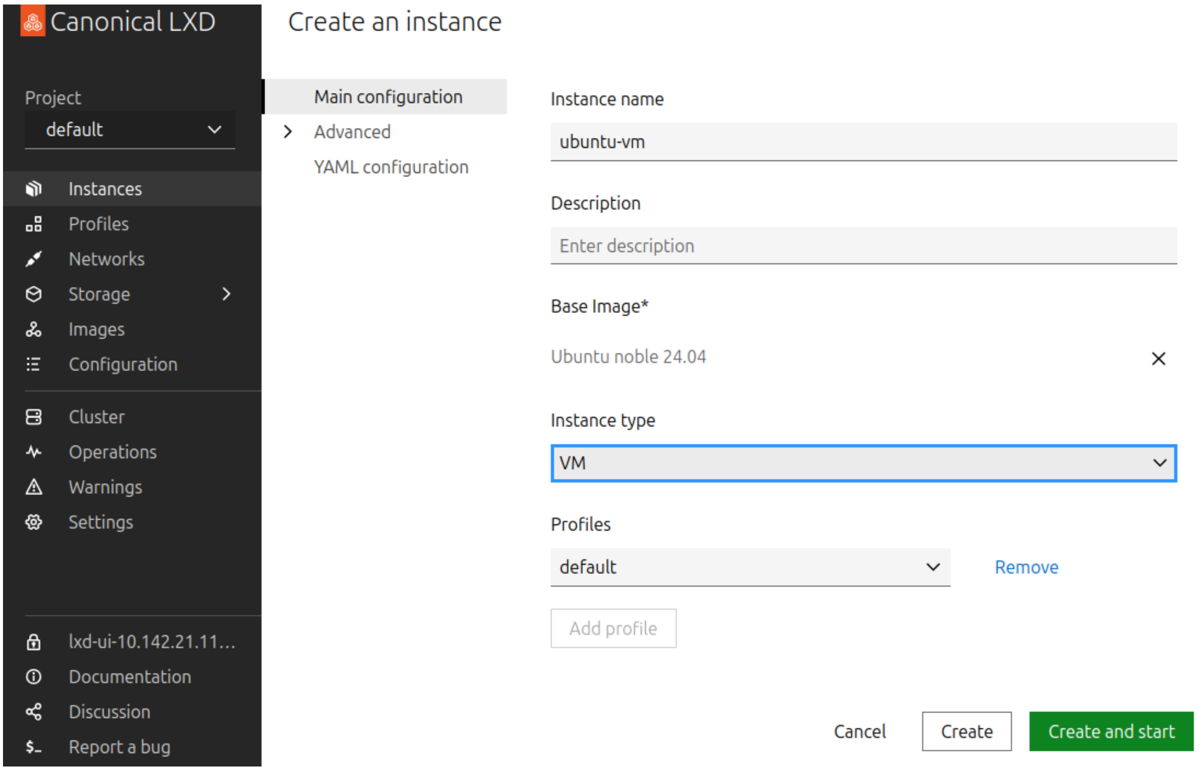
4. Create another container called `second`, using the same image. After entering the name and selecting the image, click *Create* instead of *Create and start*.

   This container will be created but not started.

> **Note**

Creating this container is quicker than launching the first, because the image is already available locally.

5. Create and start a VM called `ubuntu-vm` using the Ubuntu 24.04 LTS image. To create a VM instead of a container, select *VM* as the instance type:



> **Note**
>
> Even though you are using the same image name to launch the instance, LXD downloads a slightly different image that is compatible with VMs.

6. Start creating (do not click *Create and start* yet) a VM called `ubuntu-desktop`. When selecting the image, filter by variant "desktop" to find the Ubuntu 24.04 LTS desktop image. Note that after you select the image, the instance type is automatically set to *VM*:

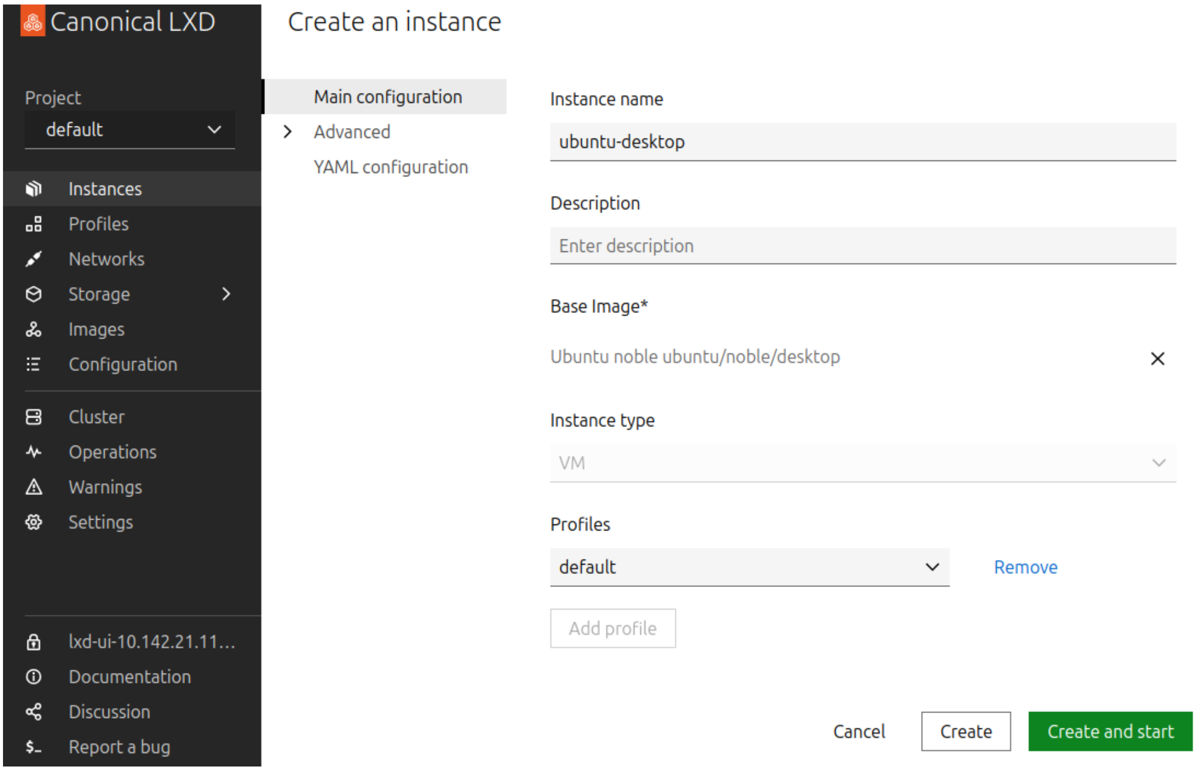
   To run smoothly, the desktop VM needs more RAM. Therefore, navigate to *Advanced > Resource limits* and set the *Memory limit* to 4 GiB. Then click *Create and start* to start the instance.

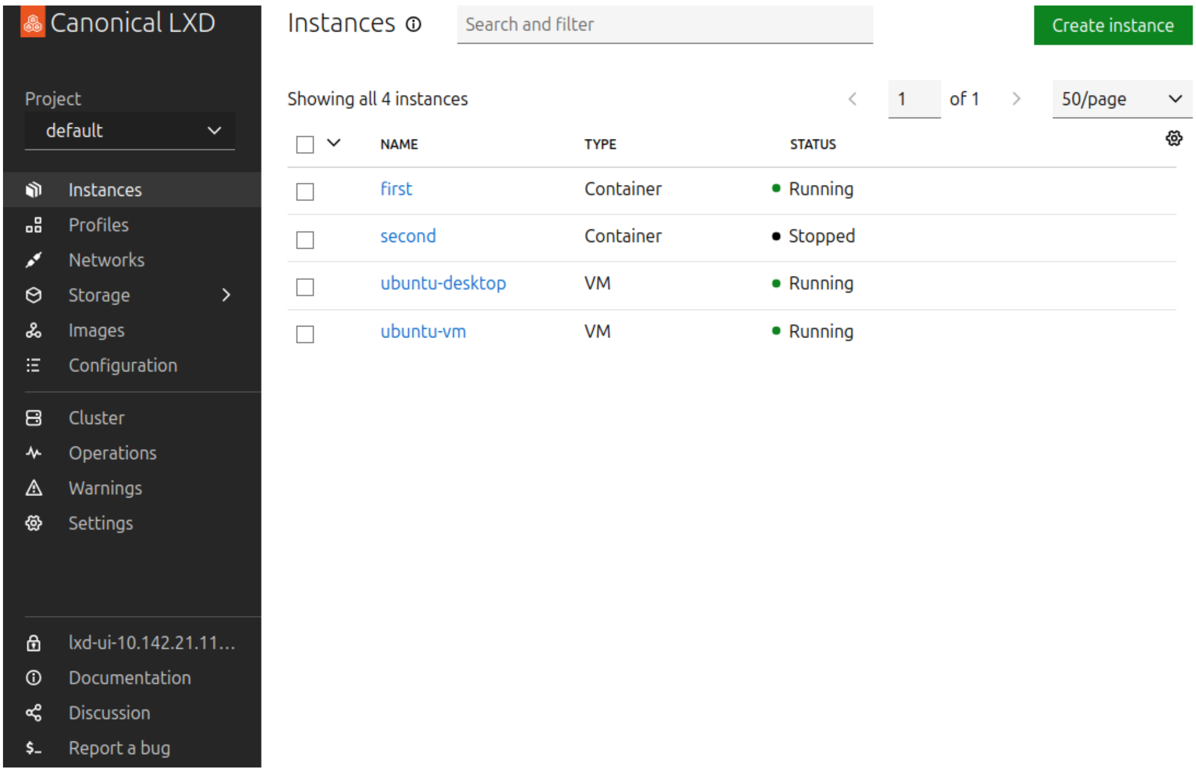
7. Check the list of instances that you created:

   You will see that all but the `second` container are running. This is because you created the `second` container but didn't start it.

   You can start the `second` container by clicking the *Start* button (□) next to it.

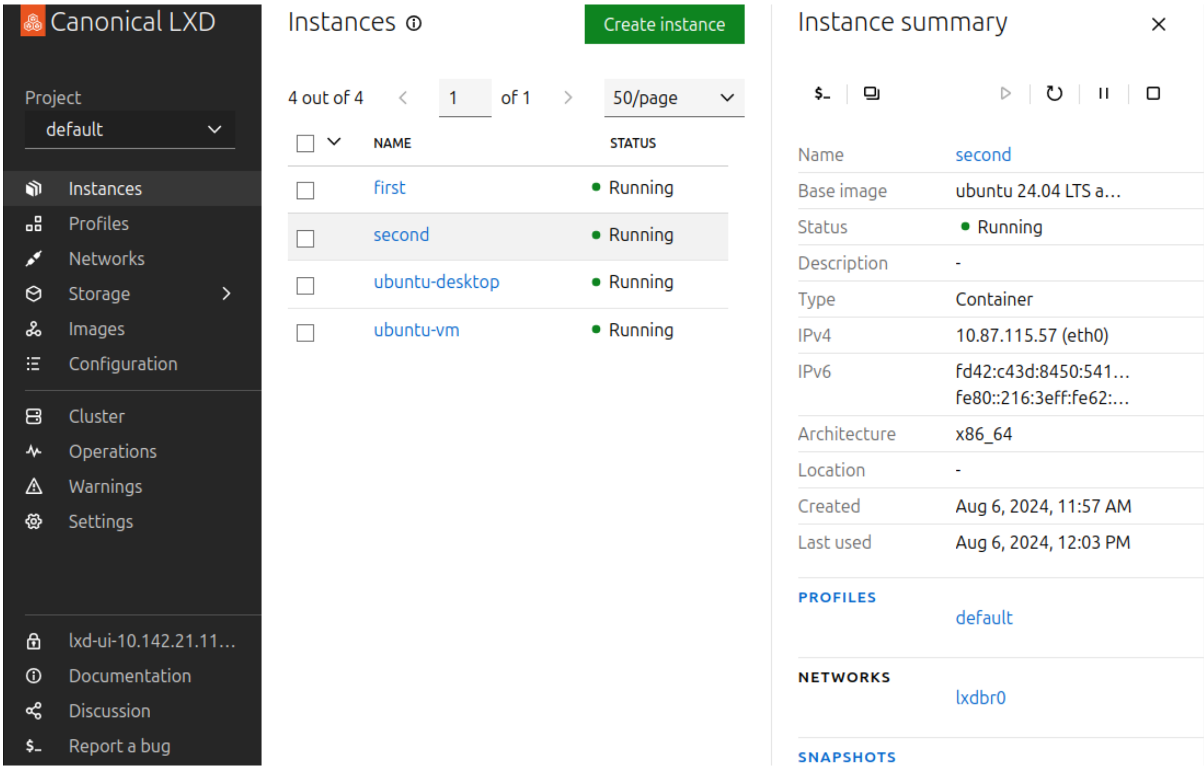
See *How to create instances* (page 73) for more information.


Canonical LXD
Create an instance
Project
default
Instances
Profiles
Networks
Storage
Images
Configuration
Cluster
Operations
Warnings
Settings
lxd-ui-10.142.21.11...
Documentation
Discussion
Report a bug
Main configuration
Advanced
YAML configuration
Instance name
ubuntu-desktop
Description
Enter description
Base Image*
Ubuntu noble ubuntu/noble/desktop
Instance type
VM
Profiles
default
Remove
Add profile
Cancel
Create
Create and start


Canonical LXD
Instances
Search and filter
Create instance
Project
default
Instances
Profiles
Networks
Storage
Images
Configuration
Cluster
Operations
Warnings
Settings
lxd-ui-10.142.21.11...
Documentation
Discussion
Report a bug
Showing all 4 instances
1
of 1
50/page
NAME
TYPE
STATUS
first
Container
Running
second
Container
Stopped
ubuntu-desktop
VM
Running
ubuntu-vm
VM
Running

1.2.4. Inspect instances

In the list of instances, click on one of the lines to see more information about the respective instance:



Click on an instance name to go to the instance detail page, where you can inspect your instance:

- The *Overview* tab shows general information and usage statistics about the instance.
- The *Configuration* tab contains the instance configuration. For now, just click through to inspect the configuration. We'll do some updates later.

  If you want to see the full instance configuration, go to the YAML configuration:
- The *Snapshots* tab shows available snapshots. You can ignore this for now; we'll look into snapshots later.
- The *Terminal* tab allows you to interact with your instance. For example, enter the following command to display information about the operating system:

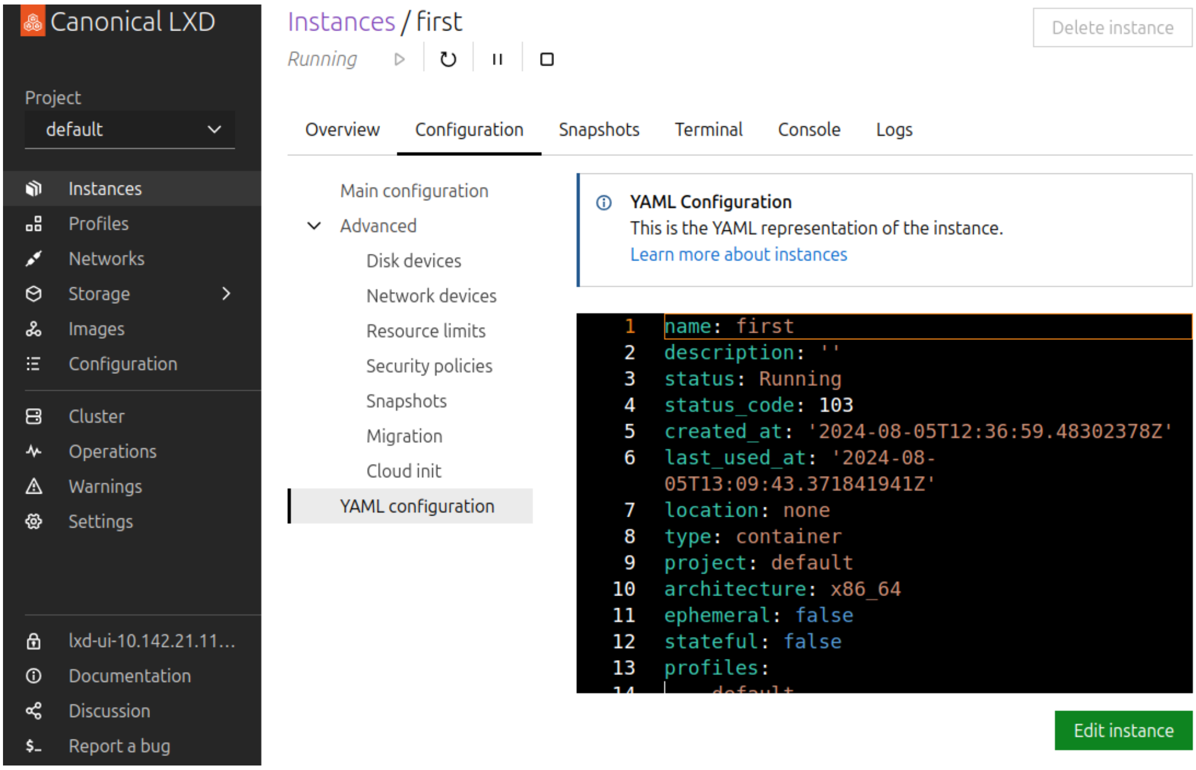
  ```
  cat /etc/*release
  ```

  Or have some fun:

  ```
  apt update
  apt install fortune
  /usr/games/fortune
  ```

  See *Get shell access to your instance* (page 115) for more information.

Note



When you navigate away from the *Terminal* tab, you are asked to confirm. The reason for this confirmation prompt is that the terminal session is not saved, so once you navigate away, your command history is lost.

- The *Console* tab is mainly relevant during startup of an instance, and for VMs.

  Go to the instance detail page of the `ubuntu-desktop` VM and check the graphic console:

  See *How to access the console* (page 109) for more information.

- The *Logs* tab contains log files for inspection and download. Running instances have only limited information. More log files are added if an instance ends up in error state.

  See *How to troubleshoot failing instances* (page 102) for more information.

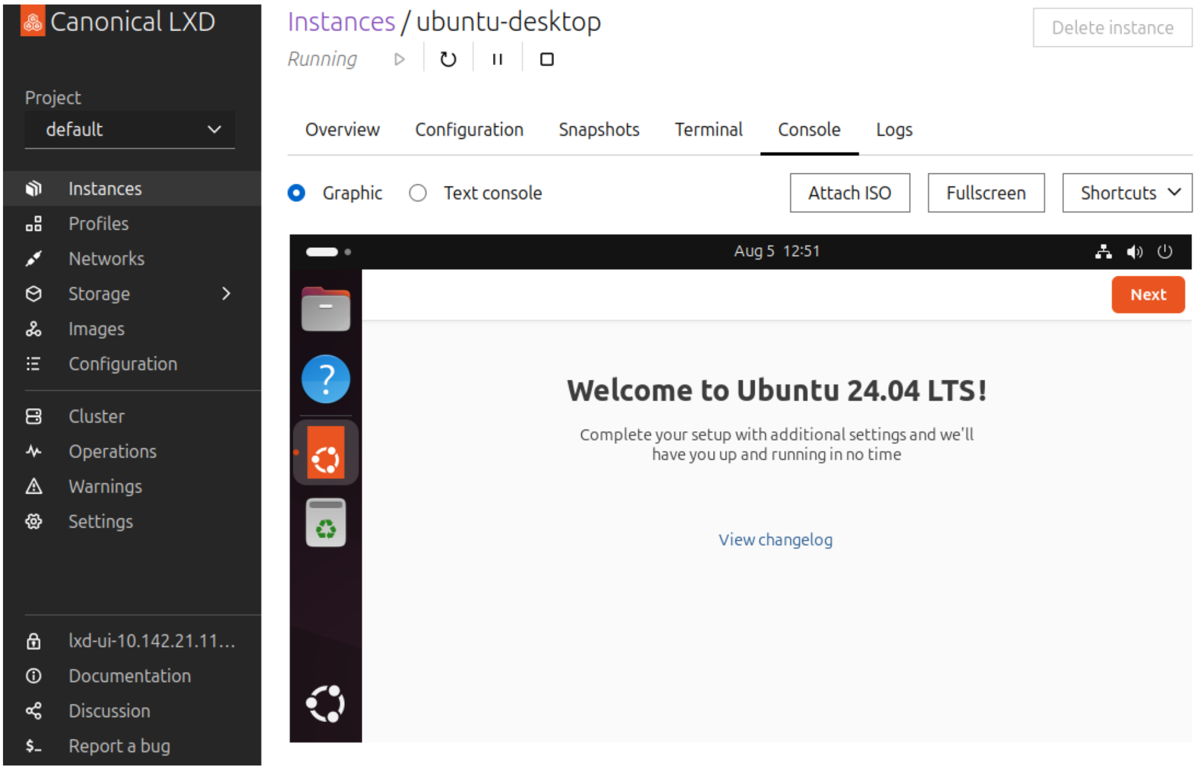
1.2.5. Stop and delete instances

For the remainder of the tutorial, we don't need all of the instances we created. So let's clean some of them up:

1. Go back to the instances list.
2. Stop the `second` container by clicking the *Stop* button (□) next to it.
3. Delete the `second` container. To do so, click the instance name to go to the instance detail page. Then click the *Delete instance* button at the top.
4. Go to the instance detail page of the `ubuntu-vm` VM to delete it.

   You will see that the *Delete instance* button at the top is not active. This is because the instance is still running.

   Stop it by clicking the *Stop* button (□) at the top, then click *Delete instance*.



> **Tip**
>
> If stopping an instance takes a long time, click the spinning *Stop* button to go back to the confirmation prompt, where you can select to force-stop the instance.

See *How to manage instances* (page 92) for more information.

1.2.6. Configure instances

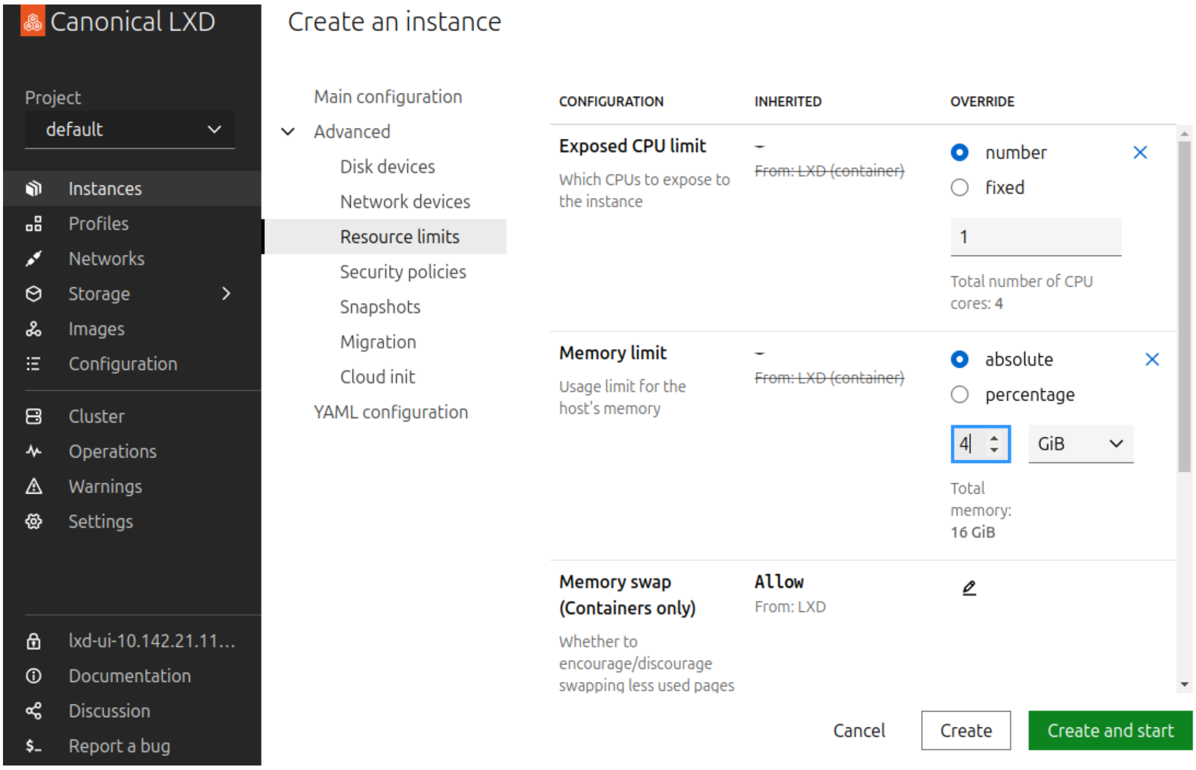
There are several limits and configuration options that you can set for your instances. See *Instance options* (page 415) for an overview.

Let's create another container with some resource limits:

1. On the instances list, click *Create instance*. Enter `limited` as the instance name and select the Ubuntu 24.04 LTS image.
2. Expand *Advanced* and go to *Resource limits*.
3. Change the *Exposed CPU limit* to 1 and the *Memory limit* to 4 GiB:
4. Click *Create and start*.
5. When the instance is running, go to its instance detail page and select *Configuration* > *Advanced* > *Resource limits*. Confirm that the limits you set are visible.
6. Go to the *Terminal* tab and enter the following commands:

   ```
   free -m
   nproc
   ```

   You should see that the total memory is limited to 4096, and the number of available CPUs is 1.



7. Go to the instance detail page of the `first` container and enter the same commands.

   You should see that the values differ from those of the `limited` container. The exact values depend on your host system (so if your host system has only one CPU, for example, you might see one CPU for the `first` container as well).

8. You can also update the configuration while your container is running:

   1. Go back to the instance detail page of the `limited` container and select *Configuration* > *Advanced* > *Resource limits*.
   2. Click *Edit instance* and change the memory limit to 2 GiB. Then save.
   3. Go to the *Terminal* tab and enter the following command:

      ```
      free -m
      ```

      Note that the number has changed.

Depending on the instance type and the storage drivers that you use, there are more configuration options that you can specify. For example, you can configure the size (quota) of the root disk device for a VM:

1. Go to the terminal for the `ubuntu-desktop` VM and check the current size of the root disk device (`/dev/sda2`):

   ```
   df -h
   ```

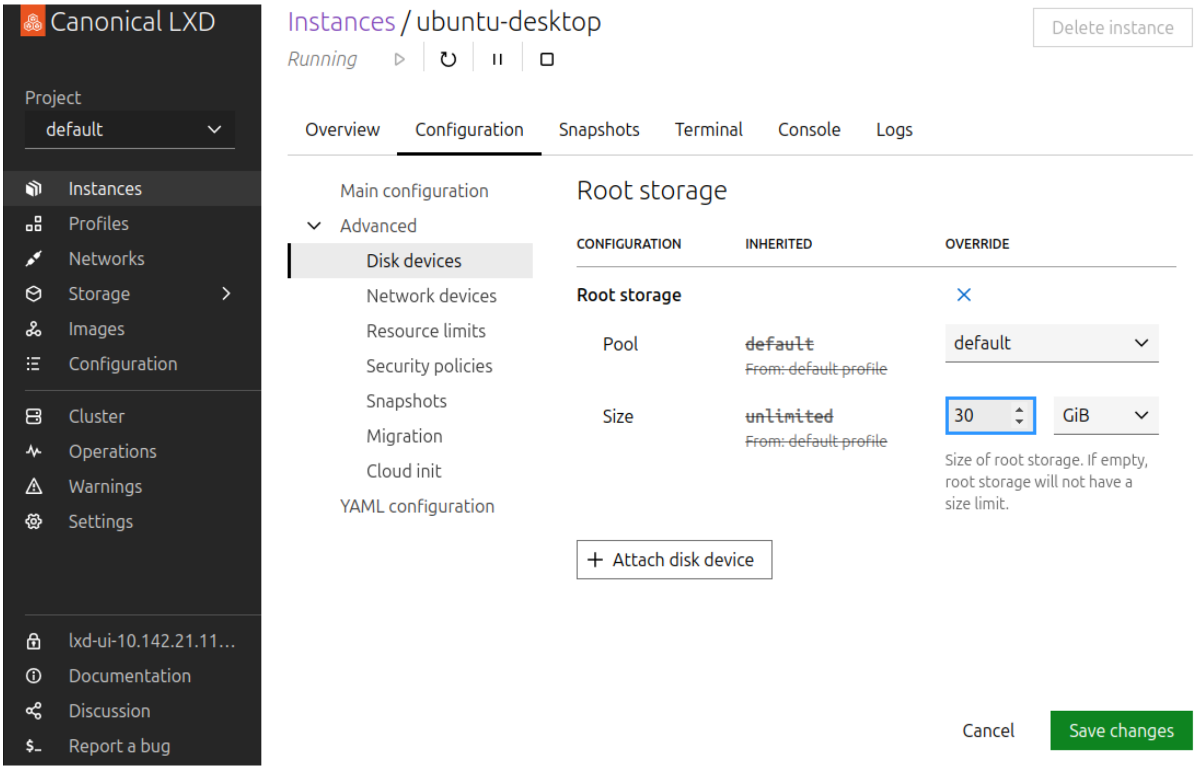
2. Navigate to *Configuration* > *Advanced* > *Disk devices*.

   Tip

By default, the size of the root disk is inherited from the default profile. Profiles store a set of configuration options and can be applied to instances instead of specifying the configuration option for each instance separately.

See *How to use profiles* (page 97) for more information.

3. Override the size of the root disk device to be 30 GiB:



4. Save the configuration, and then restart the VM by clicking the *Restart* button ().

5. In the terminal, check the size of the root disk device again:

```
df -h
```

   Note that the size has changed.

See *How to configure instances* (page 84) and *Instance configuration* (page 414) for more information.

1.2.7. Manage snapshots

You can create a snapshot of your instance, which makes it easy to restore the instance to a previous state.

1. Go to the instance detail page of the `first` container and select the *Snapshots* tab.

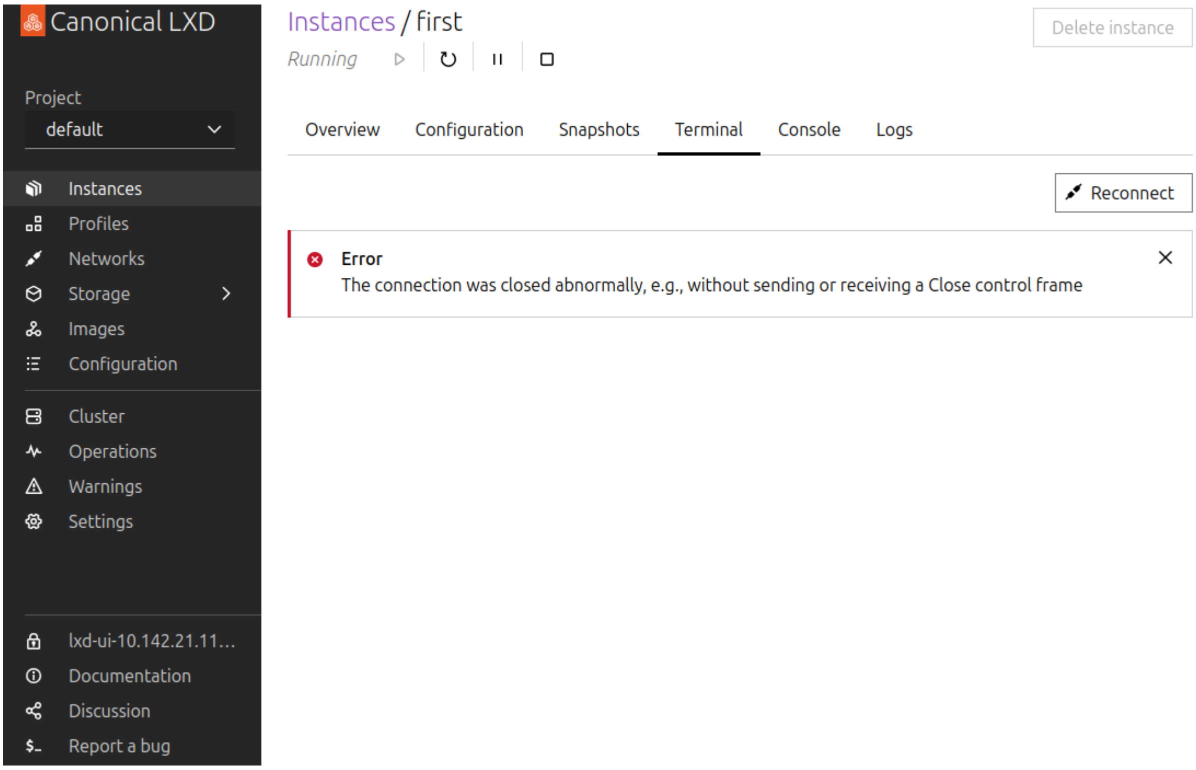
2. Click *Create snapshot* and enter the snapshot name `clean`. Leave the other options unchanged and create the snapshot.

   You should now see the snapshot in the list.

3. Go to the *Terminal* tab and break the container:

```
rm /usr/bin/bash
```

4. Confirm the breakage by reloading the page:



   The UI cannot open a terminal in your container anymore, because you deleted the `bash` command.

5. Go back to the *Snapshots* tab and restore the container to the state of the snapshot by clicking the *Restore snapshot* button () next to it.
6. Go back to the *Terminal* tab. The terminal should now load again.
7. Go to the *Snapshots* tab and delete the snapshot by clicking the *Delete snapshot* button () next to it.

See *Use snapshots for instance backup* (page 128) for more information.

1.2.8. Add a custom storage volume

You can add additional storage to your instance, and also share storage between different instances. See *Storage volumes* (page 352) for more information.

Let's start by creating a custom storage volume:

1. Navigate to *Storage* > *Volumes*.

   Even though you have not created any custom storage volumes yet, you should see several storage volumes in the list. These are instance volumes (which contain the root disks of your instances) and image volumes (which contain the cached base images).

2. Click `Create volume` and enter a name and a size. Leave the default content type (`filesystem`).

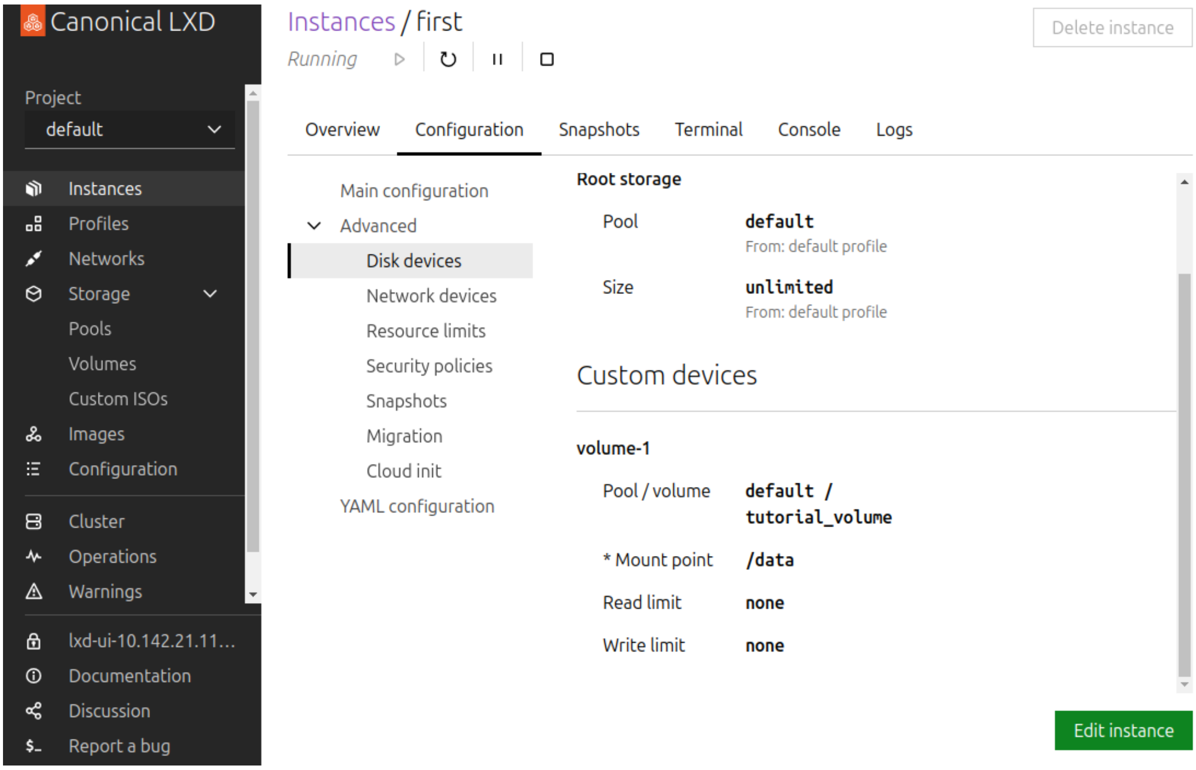
After creating the instance, we can attach it to some instances:

1. Go to the instance detail page of the `first` container.

2. Go to *Configuration* > *Advanced* > *Disk devices*.
3. Click *Edit instance* and then *Attach disk device*.
4. Select the disk device that you just created.

> **Tip**
>
> You can create a custom volume directly from this screen as well.

5. Enter `/data` as the mount point and save your changes:



6. Go to the *Terminal* tab and enter the following command to create a file on the custom volume:

```
touch /data/hello_world
```

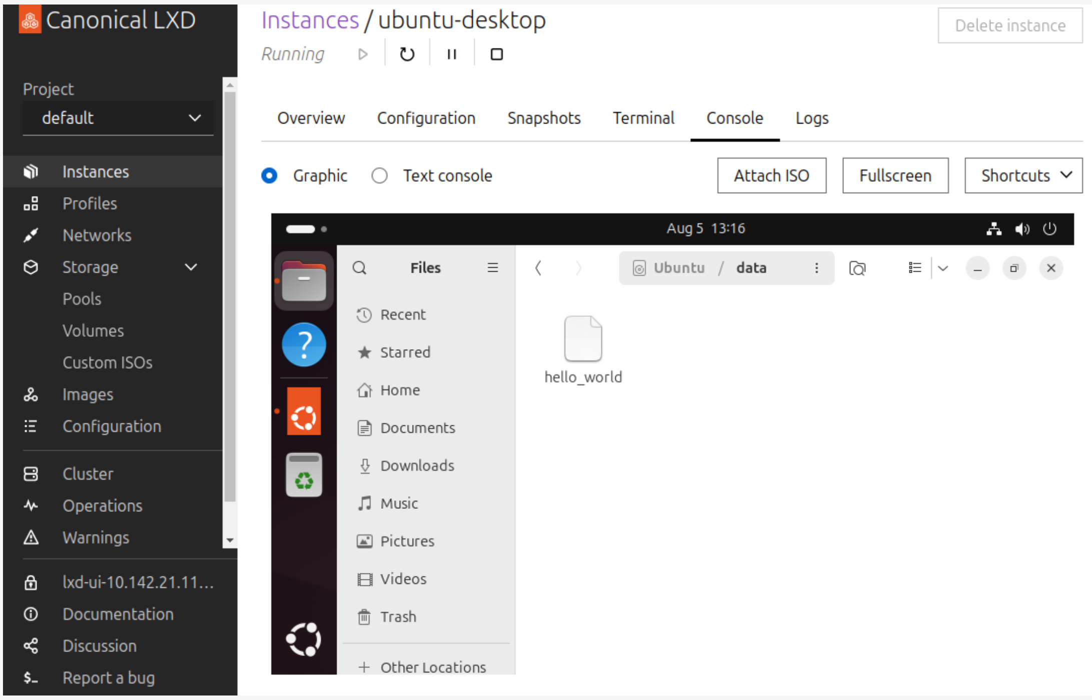
7. Go to the instance detail page of the `ubuntu-desktop` VM and add the same custom storage volume with the same mount point (`/data`).
8. Go to the *Terminal* tab and enter the following command to see the file you created from your `first` container:

```
ls /data/
```

> **Note**
>
> You can also look at the directory in the file browser. To do so, enter the following command in the terminal first:

```
chown ubuntu /data /data/*
```

Then switch to the console, open the file browser, and navigate to the `/data` folder:



1.2.9. Use projects

You can use projects to group related instances, and other entities, on your LXD server. See *Instances grouping with projects* (page 368) and *How to create and configure projects* (page 161) for more information.

Originally, there is only a default project on the server. All the instances you created so far are part of this project.

Now, let's create another project:

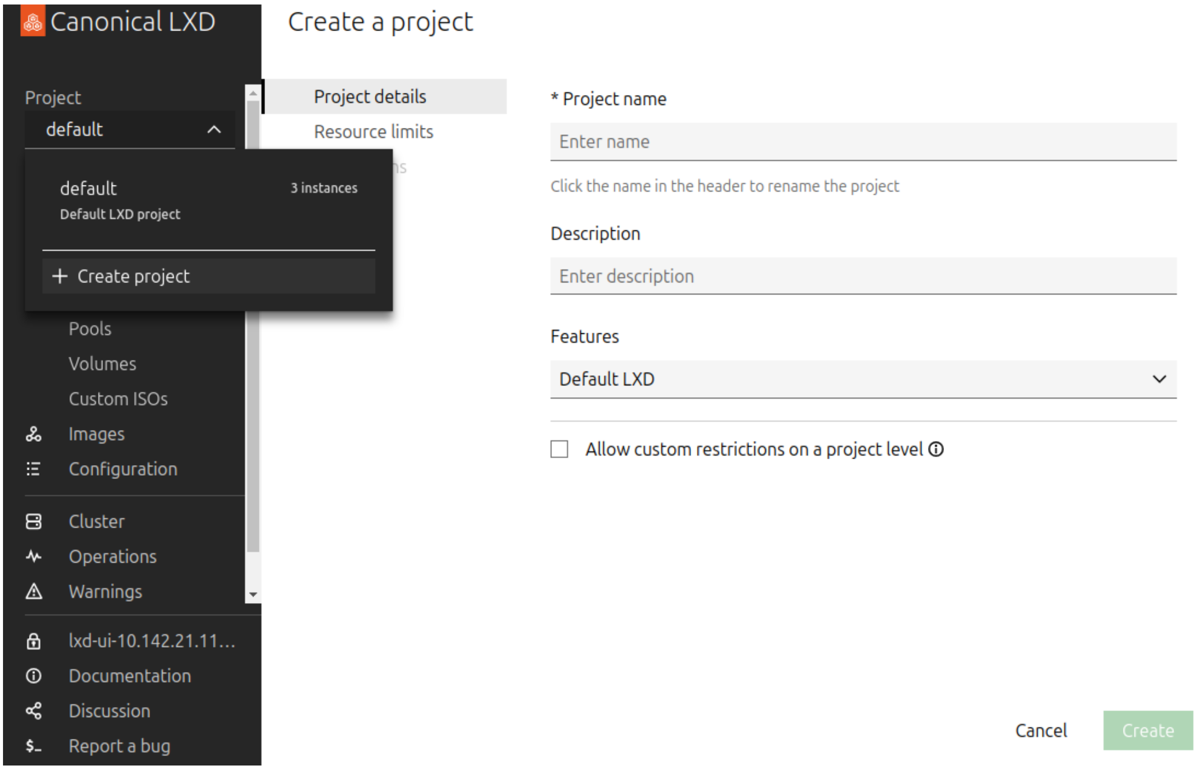
1. Expand the *Project* dropdown and click *Create project*:
2. Enter `tutorial` as the project name.
3. For features, select `Customised`.

   You can then select which features should be isolated. "Isolated" in this context means that if you select one of the features, entities of this type are confined to the project. So when you use a project where, for example, storage volumes are isolated, you can see only the storage volumes that are defined within the project.
4. Deselect `Storage volumes` and create the project.

The new project is automatically selected for you. Let's check its content:

1. Go to *Instances*.

   You will see that there are no instances in your project, because instances are always isolated, and the instances you created earlier are in the default project.
2. Create an instance in the new project.



You should notice that you get an error about missing root storage. The reason for this is that the root storage is usually defined in the default profile, but profiles are isolated in the project.

3. To resolve the error, edit the root storage. Use the `default` pool and leave the size empty.

   Then create the instance.

4. Go to *Storage* > *Volumes*.

   Remember that in the default project, you saw three different kinds of storage volumes in the volume list:

   - Instance (container or VM) volumes
   - Image volumes
   - Custom volumes

   You should see the same three types in the `tutorial` project. However, note the following:

   - You can see only one instance volume and one image volume. These are for the one instance you created in the `tutorial` project.

     You cannot see the instance and image volumes for the instances you created in the default project, because both instances and images are isolated in the `tutorial` project, so you cannot see the corresponding storage volumes from other projects.

   - You can see the custom storage volume that you created in the default project.

     Because you deselected `Storage volumes` when creating the project, storage volumes are not isolated, and you can therefore see storage volumes from other projects.

1.2.10. Clean up entities

Now that we've run through the basic functionality of LXD, let's clean up the entities we created throughout the tutorial.

1. With the `tutorial` project still selected, go to the instances list and stop and delete the instance you created in this project.
2. Go to *Images* and click the *Delete* button () next to it.
3. Go to *Configuration* and click the *Delete project* button in the top-right corner.

   After deleting the project, you are automatically switched back to the default project.
4. Stop and delete all instances in the default project. To do this all at once, go to the instance list and select all instances. Then click *Stop* at the top. Finally, click *Delete* at the top.
5. Go to *Storage* > *Volumes* and click the *Delete* button () next to the `tutorial_volume` storage volume.

> **Note**
>
> Optionally, you can also delete the images that you used. However, this isn't really needed. If you keep them, they will eventually expire (by default, when they haven't been used for ten days).

1.2.11. Next steps

Now that you've done your first experiments with LXD, you should read up on important concepts in the *Explanation* (page 345) section and check out the *How-to guides* (page 28) to start working with LXD!

2. How-to guides

These how-to guides cover key operations and processes in LXD.

2.1. Get started

To get started with LXD, install and initialize it. Then do some basic configuration of the server and the command-line client.

2.1.1. Getting started

To get started with LXD, see the documentation in this section.

How to install and initialize LXD:

How to install LXD

There are multiple approaches to installing LXD, depending on your Linux distribution, operating system, and use case.

Install the LXD snap package

The recommended way to install LXD is its snap package[26], available for many Linux distributions. For alternative methods, see: *Other Linux installation options* (page 29), *Other operating systems* (page 30), or *Install LXD from source* (page 31).

Requirements

- The LXD snap must be available for your Linux distribution[27].
- The `snapd` daemon[28] must be installed.

Install

Use this command to install LXD from the recommended *default snap track* (page 390) (currently 5.21):

```
sudo snap install lxd
```

If you are installing LXD on a *cluster member* (page 370), add the `--cohort="+"` flag to *keep cluster members synchronized* (page 323) to the same snap version:

```
sudo snap install lxd --cohort="+"
```

Next, follow the *Post-installation* (page 29) steps below.

[26] https://snapcraft.io/lxd
[27] https://snapcraft.io/lxd#distros
[28] https://snapcraft.io/docs/installing-snapd

Optionally specify a channel

Channels correspond to different *LXD releases* (page 388). When unspecified, the LXD snap defaults to the most recent `stable` LTS, which is recommended for most use cases.

To specify a different channel, add the `--channel` flag at installation:

```
sudo snap install lxd --channel=<target channel> [--cohort="+"]
```

For example, to use the `6/stable` channel, run:

```
sudo snap install lxd --channel=6/stable
```

For details about LXD snap channels, see: *Channels* (page 389).

Post-installation

Follow these steps after installing the LXD snap.

Add the current user

To allow the current user to interact with the LXD daemon, update the `lxd` group:

```
getent group lxd | grep -qwF "$USER" || sudo usermod -aG lxd "$USER"
```

Afterward, apply the change to your current shell session by running:

```
newgrp lxd
```

For more information, see the *Manage access to LXD* (page 34) section below.

Hold or schedule updates

When a new release is published to a snap channel, installed snaps following that channel update automatically by default.

For *LXD clusters* (page 370), or on any machine where you want control over updates, you should override this default behavior by either holding or scheduling updates. See: *Manage updates* (page 321).

Other Linux installation options

Some Linux installations can use package managers other than Snap to install LXD. These managers all install the latest *feature release* (page 388).

Alpine Linux

Arch Linux

Fedora

Gentoo

Run:

```
apk add lxd
```

Run:

```
pacman -S lxd
```

Fedora RPM packages for LXC/LXD are available in the COPR repository[29]. These are unofficial and minimally tested; use at your own risk.

View the installation guide[30] for details.

Run:

```
emerge --ask lxd
```

Following installation, make sure to *manage access to LXD* (page 34).

Other operating systems

Builds of the *lxc* (page 690) client are available for non-Linux operating systems to enable interaction with remote LXD servers. For more information, see: *About lxd and lxc* (page 345).

macOS

Windows

The Homebrew[31] package manager must be installed on your system.

To install the client from the latest *feature release* (page 388) of LXD, run:

```
brew install lxc
```

The Chocolatey[32] package manager must be installed on your system.

To install the client from the latest *feature release* (page 388) of LXD, run:

```
choco install lxc
```

Native builds of the client

You can find native builds of the *lxc* (page 690) client on GitHub[33]:

- Linux: `bin.linux.lxc.aarch64`[34], `bin.linux.lxc.x86_64`[35]
- Windows: `bin.windows.lxc.aarch64.exe`[36], `bin.windows.lxc.x86_64.exe`[37]
- macOS: `bin.macos.lxc.aarch64`[38], `bin.macos.lxc.x86_64`[39]

[29] https://copr.fedorainfracloud.org/coprs/ganto/lxc4/
[30] https://github.com/ganto/copr-lxc4/wiki
[31] https://brew.sh
[32] https://chocolatey.org
[33] https://github.com/canonical/lxd
[34] https://github.com/canonical/lxd/releases/latest/download/bin.linux.lxc.aarch64
[35] https://github.com/canonical/lxd/releases/latest/download/bin.linux.lxc.x86_64
[36] https://github.com/canonical/lxd/releases/latest/download/bin.windows.lxc.aarch64.exe
[37] https://github.com/canonical/lxd/releases/latest/download/bin.windows.lxc.x86_64.exe
[38] https://github.com/canonical/lxd/releases/latest/download/bin.macos.lxc.aarch64
[39] https://github.com/canonical/lxd/releases/latest/download/bin.macos.lxc.x86_64

To download a specific build:

1. Make sure that you are logged into your GitHub account.
2. Filter for the branch or tag that you are interested in (for example, the latest release tag or `main`).
3. Select the latest build and download the suitable artifact.

These builds are for the *lxc* (page 690) client only, not the LXD daemon. For an explanation of the differences, see: *About lxd and lxc* (page 345).

Install LXD from source

These instructions for building and installing from source are suitable for developers who want to build the latest version of LXD, or to build a specific release of LXD which may not be offered by their Linux distribution. Source builds for integration into Linux distributions are not covered.

We recommend having the latest versions of `liblxc` (see *LXC requirements* (page 385)) available for LXD development. For convenience, `make deps` will pull the appropriate versions of `liblxc` and `dqlite` from their corresponding upstream Git repository. Additionally, LXD requires a modern Golang (see *Go* (page 385)) version to work. On Ubuntu, you can install these with:

```
sudo apt update
sudo apt install \
    autoconf \
    automake \
    build-essential \
    gettext \
    git \
    libacl1-dev \
    libapparmor-dev \
    libcap-dev \
    liblz4-dev \
    libseccomp-dev \
    libsqlite3-dev \
    libtool \
    libudev-dev \
    libuv1-dev \
    make \
    meson \
    ninja-build \
    pkg-config \
    python3-venv
command -v snap >/dev/null || sudo apt-get install snapd
sudo snap install --classic go
```

Note

If you use the `liblxc-dev` package and get compile time errors when building the `go-lxc` module, ensure that the value for `LXC_DEVEL` is `0` for your `liblxc` build. To check this, look at

/usr/include/lxc/version.h. If the LXC_DEVEL value is 1, replace it with 0 to work around the problem. It's a packaging bug that is now fixed, see LP: #2039873[40].

There are a few storage drivers for LXD besides the default dir driver. Installing these tools adds a bit to initramfs and may slow down your host boot, but are needed if you'd like to use a particular driver:

```
sudo apt install lvm2 thin-provisioning-tools
sudo apt install btrfs-progs
```

At runtime, LXD might need the following packages to be installed on the host system:

```
sudo apt update
sudo apt install \
    attr \
    iproute2 \
    nftables \
    rsync \
    squashfs-tools \
    tar \
    xz-utils

# `nftables` can be replaced by `iptables` on older systems
```

To run the test suite or test related make targets, you'll also need:

```
sudo apt update
sudo apt install \
    acl \
    bind9-dnsutils \
    btrfs-progs \
    busybox-static \
    curl \
    dnsmasq-base \
    dosfstools \
    e2fsprogs \
    iputils-ping \
    jq \
    netcat-openbsd \
    openvswitch-switch \
    s3cmd \
    shellcheck \
    socat \
    sqlite3 \
    swtpm \
    xfsprogs \
    yq
```

[40] https://bugs.launchpad.net/ubuntu/+source/lxc/+bug/2039873

From source: Build the latest version

These instructions for building from source are suitable for individual developers who want to build the latest version of LXD, or build a specific release of LXD which may not be offered by their Linux distribution. Source builds for integration into Linux distributions are not covered here and may be covered in detail in a separate document in the future.

```
git clone https://github.com/canonical/lxd
cd lxd
```

This will download the current development tree of LXD and place you in the source tree. Then proceed to the instructions below to actually build and install LXD.

From source: Build a release

The LXD release tarballs bundle a complete dependency tree as well as a local copy `libdqlite` for LXD's database setup.

```
tar zxvf lxd-4.18.tar.gz
cd lxd-4.18
```

This will unpack the release tarball and place you inside of the source tree. Then proceed to the instructions below to actually build and install LXD.

Start the build

The actual building is done by two separate invocations of the Makefile: `make deps` – which builds libraries required by LXD – and `make`, which builds LXD itself. At the end of `make deps`, a message will be displayed which will specify environment variables that should be set prior to invoking `make`. As new versions of LXD are released, these environment variable settings may change, so be sure to use the ones displayed at the end of the `make deps` process, as the ones below (shown for example purposes) may not exactly match what your version of LXD requires:

We recommend having at least 2GiB of RAM to allow the build to complete.

```
~$ make deps ...make[1]: Leaving directory '/root/go/deps/dqlite'#
environment Please set the following in your environment (possibly
~/.bashrc)# export CGO_CFLAGS="${CGO_CFLAGS} -I$(go env
GOPATH)/deps/dqlite/include/"# export CGO_LDFLAGS="${CGO_LDFLAGS} -L$(go env
GOPATH)/deps/dqlite/.libs/"# export LD_LIBRARY_PATH="$(go env
GOPATH)/deps/dqlite/.libs/${LD_LIBRARY_PATH}"# export
CGO_LDFLAGS_ALLOW="(-Wl,-wrap,pthread_create)|(-Wl,-z,now)" ~$ make
```

From source: Install

Once the build completes, you simply keep the source tree, add the directory referenced by `$(go env GOPATH)/bin` to your shell path, and set the `LD_LIBRARY_PATH` variable printed by `make deps` to your environment. This might look something like this for a `~/.bashrc` file:

```
export PATH="${PATH}:$(go env GOPATH)/bin"
export LD_LIBRARY_PATH="$(go env GOPATH)/deps/dqlite/.libs/:${LD_LIBRARY_PATH}"
```

Now, the `lxd` and `lxc` binaries will be available to you and can be used to set up LXD. The binaries will automatically find and use the dependencies built in `$(go env GOPATH)/deps` thanks to the `LD_LIBRARY_PATH` environment variable.

Machine setup

You'll need sub{u,g}ids for root, so that LXD can create the unprivileged containers:

```
echo "root:1000000:1000000000" | sudo tee -a /etc/subuid /etc/subgid
```

By default, only users added to the `lxd` group can interact with the LXD daemon. Installing from source doesn't guarantee that the `lxd` group exists in the system. If you want the current user (or any other user) to be able to interact with the LXD daemon, create the group and add the user to it:

```
getent group lxd >/dev/null || sudo groupadd --system lxd # create the group if needed
getent group lxd | grep -qwF "$USER" || sudo usermod -aG lxd "$USER"
```

Afterward, apply the change to your current shell session by running:

```
newgrp lxd
```

Now you can run the daemon (the `--group sudo` bit allows everyone in the `sudo` group to talk to LXD; you can create your own group if you want):

```
sudo -E PATH=${PATH} LD_LIBRARY_PATH=${LD_LIBRARY_PATH} $(go env GOPATH)/bin/lxd --group sudo
```

Note

If `newuidmap/newgidmap` tools are present on your system and `/etc/subuid`, `etc/subgid` exist, they must be configured to allow the root user a contiguous range of at least 10M UID/GID.

Shell completions

Shell completion profiles can be generated with `lxc completion <shell>` (e.g. `lxc completion bash`). Supported shells are `bash`, `zsh`, `fish`, and `powershell`.

```
lxc completion bash > /etc/bash_completion.d/lxc # generating completions for bash as an example
. /etc/bash_completion.d/lxc
```

Manage access to LXD

Access control for LXD is based on group membership. The root user and all members of the `lxd` group can interact with the local daemon.

On Ubuntu images, the `lxd` group already exists and the root user is automatically added to it. The group is also created during installation if you *installed LXD from the snap* (page 28).

To check if the `lxd` group exists, run:

```
getent group lxd
```

If this command returns no result, the `lxd` group is missing from your system. This might be the case if you *installed LXD from source* (page 31). To create the group and restart the LXD daemon, run:

```
getent group lxd >/dev/null || sudo groupadd --system lxd
```

Afterward, add trusted users to the group so they can use LXD. The following command adds the current user:

```
getent group lxd | grep -qwF "$USER" || sudo usermod -aG lxd "$USER"
```

Afterward, apply the change to your current shell session by running:

```
newgrp lxd
```

Important security notice

Local access to LXD through the Unix socket always grants full access to LXD. This includes the ability to attach file system paths or devices to any instance as well as tweak the security features on any instance.

Therefore, you should only give such access to users who you'd trust with root access to your system.

For more information, see *Access to the LXD daemon* (page 377).

Upgrade LXD

After upgrading LXD to a newer version, LXD might need to update its database to a new schema. This update happens automatically when the daemon starts up after a LXD upgrade. A backup of the database before the update is stored in the same location as the active database (for example, at `/var/snap/lxd/common/lxd/database` for the snap installation).

Important

After a schema update, older versions of LXD might regard the database as invalid. That means that downgrading LXD might render your LXD installation unusable.

In that case, if you need to downgrade, restore the database backup before starting the downgrade.

How to initialize LXD

Before you can create a LXD instance, you must configure and initialize LXD.

Interactive configuration

Run the following command to start the interactive configuration process:

```
lxd init
```

Note

For simple configurations, you can run this command as a normal user. However, some more advanced operations during the initialization process (for example, joining an existing cluster) require root privileges. In this case, run the command with sudo or as root.

The tool asks a series of questions to determine the required configuration. The questions are dynamically adapted to the answers that you give. They cover the following areas:

Clustering (see Clusters (page 370) and How to form a cluster (page 280))
A cluster combines several LXD servers. The cluster members share the same distributed database and can be managed uniformly using the LXD client (*lxc* (page 690)) or the REST API.

The default answer is no, which means clustering is not enabled. If you answer yes, you can either connect to an existing cluster or create one.

MAAS support (see maas.io[41] and MAAS - Setting up LXD for VMs[42])
MAAS is an open-source tool that lets you build a data center from bare-metal servers.

The default answer is no, which means MAAS support is not enabled. If you answer yes, you can connect to an existing MAAS server and specify the name, URL and API key.

Networking (see Networking setups (page 353) and Network devices (page 449))
Provides network access for the instances.

You can let LXD create a new bridge (recommended) or use an existing network bridge or interface.

You can create additional bridges and assign them to instances later.

Storage pools (see Storage pools, volumes, and buckets (page 349) and Storage drivers (page 521))
Instances (and other data) are stored in storage pools.

For testing purposes, you can create a loop-backed storage pool. For production use, however, you should use an empty partition (or full disk) instead of loop-backed storage (because loop-backed pools are slower and their size/quota can't be reduced).

The recommended backends are zfs and btrfs.

You can create additional storage pools later.

Remote access (see Access to the remote API (page 377) and Remote API authentication (page 358))
Allows remote access to the server over the network.

The default answer is no, which means remote access is not allowed. If you answer yes, you can connect to the server over the network.

41 https://maas.io/
42 https://maas.io/docs/how-to-manage-machines#p-9078-use-lxd-vms

You can choose to add client certificates to the server either manually or through tokens.

Automatic image update (see Local and remote images (page 348))
You can download images from image servers. In this case, images can be updated automatically.

The default answer is `yes`, which means that LXD will update the downloaded images regularly.

YAML `lxd init` preseed (see Non-interactive configuration (page 37))
If you answer `yes`, the command displays a summary of your chosen configuration options in the terminal.

Minimal setup

To create a minimal setup with default options, you can skip the configuration steps by adding the `--minimal` flag to the `lxd init` command:

```
lxd init --minimal
```

Note

The minimal setup provides a basic configuration, but the configuration is not optimized for speed or functionality. Especially the *dir storage driver* (page 553), which is used by default, is slower than other drivers and doesn't provide fast snapshots, fast copy/launch, quotas and optimized backups.

If you want to use an optimized setup, go through the interactive configuration process instead.

Non-interactive configuration

The `lxd init` command supports a `--preseed` command line flag that makes it possible to fully configure the LXD daemon settings, storage pools, network devices and profiles, in a non-interactive way through a preseed YAML file.

For example, starting from a brand new LXD installation, you could configure LXD with the following command:

```
    cat <<EOF | lxd init --preseed
config:
  core.https_address: 192.0.2.1:9999
  images.auto_update_interval: 15
networks:
- name: lxdbr0
  type: bridge
  config:
    ipv4.address: auto
    ipv6.address: none
EOF
```

This preseed configuration initializes the LXD daemon to listen for HTTPS connections on port 9999 of the 192.0.2.1 address, to automatically update images every 15 hours and to create a network bridge device named `lxdbr0`, which gets assigned an IPv4 address automatically.

Re-configuring an existing LXD installation

If you are configuring a new LXD installation, the preseed command applies the configuration as specified (as long as the given YAML contains valid keys and values). There is no existing state that might conflict with the specified configuration.

However, if you are re-configuring an existing LXD installation using the preseed command, the provided YAML configuration might conflict with the existing configuration. To avoid such conflicts, the following rules are in place:

- The provided YAML configuration overwrites existing entities. This means that if you are re-configuring an existing entity, you must provide the full configuration for the entity and not just the different keys.
- If the provided YAML configuration contains entities that do not exist, they are created.

This is the same behavior as for a `PUT` request in the *REST API* (page 618).

Rollback

If some parts of the new configuration conflict with the existing state (for example, they try to change the driver of a storage pool from `dir` to `zfs`), the preseed command fails and automatically attempts to roll back any changes that were applied so far.

For example, it deletes entities that were created by the new configuration and reverts overwritten entities back to their original state.

Failure modes when overwriting entities are the same as for the `PUT` requests in the *REST API* (page 618).

> **Note**
>
> The rollback process might potentially fail, although rarely (typically due to backend bugs or limitations). You should therefore be careful when trying to reconfigure a LXD daemon via preseed.

Default profile

Unlike the interactive initialization mode, the `lxd init --preseed` command does not modify the default profile, unless you explicitly express that in the provided YAML payload.

For instance, you will typically want to attach a root disk device and a network interface to your default profile. See the following section for an example.

Configuration format

The supported keys and values of the various entities are the same as the ones documented in the *REST API* (page 618), but converted to YAML for convenience. However, you can also use JSON, since YAML is a superset of JSON.

The following snippet gives an example of a preseed payload that contains most of the possible configurations. You can use it as a template for your own preseed file and add, change or remove what you need:

```yaml
# Daemon settings
config:
  core.https_address: 192.0.2.1:9999
  images.auto_update_interval: 6

# Storage pools
storage_pools:
- name: data
  driver: zfs
  config:
    source: my-zfs-pool/my-zfs-dataset

# Storage volumes
storage_volumes:
- name: my-vol
  pool: data

# Network devices
networks:
- name: lxd-my-bridge
  type: bridge
  config:
    ipv4.address: auto
    ipv6.address: none

# Profiles
profiles:
- name: default
  devices:
    root:
      path: /
      pool: data
      type: disk
- name: test-profile
  description: "Test profile"
  config:
    limits.memory: 2GiB
  devices:
    test0:
      name: test0
      nictype: bridged
```

(continues on next page)

```
    parent: lxd-my-bridge
    type: nic
```

See *Preseed YAML file fields* (page 508) for the complete fields of the preseed YAML file.

How to access the UI and the local, offline copy of the documentation:

How to access the LXD web UI

Note

The LXD web UI is available as part of the LXD snap.

See the LXD-UI GitHub repository[43] for the source code.

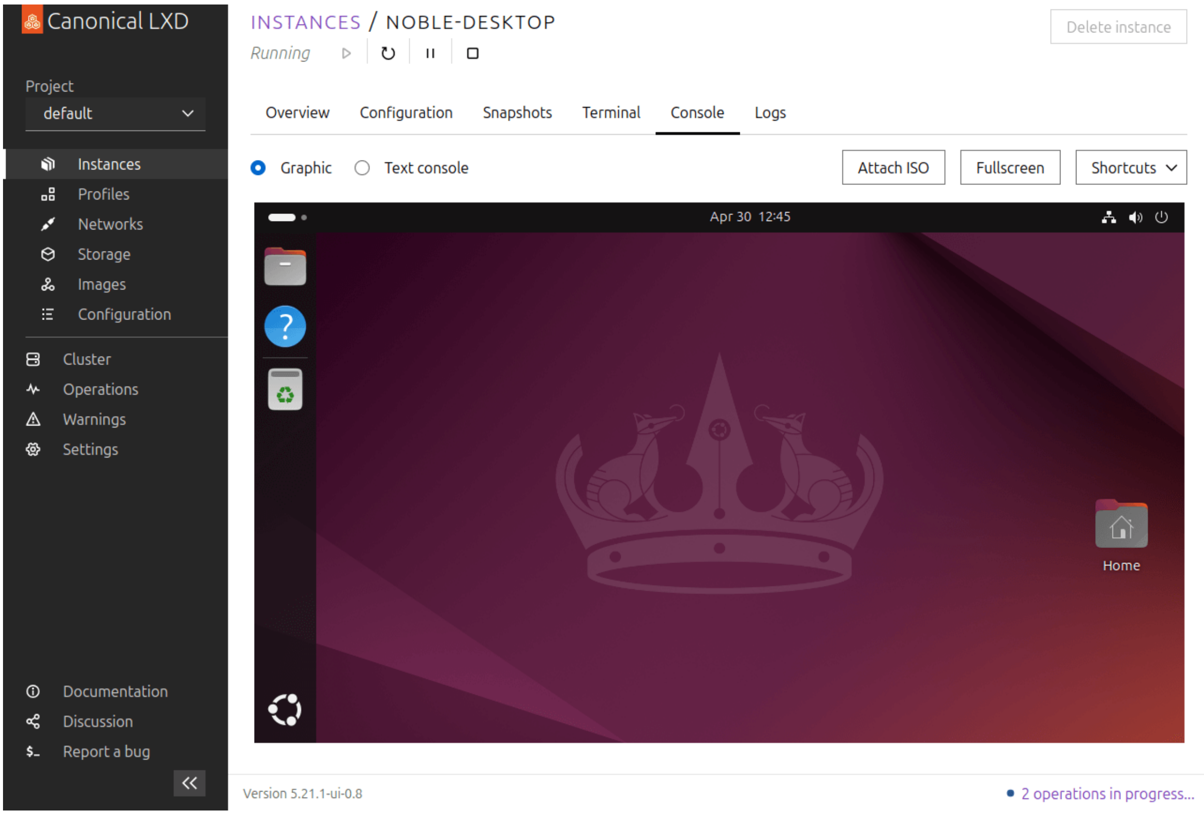


Fig. 1: Graphical console of an instance in the LXD web UI

The LXD web UI provides you with a graphical interface to manage your LXD server and instances. It does not provide full functionality yet, but it is constantly evolving, already covering many of the features of the LXD command-line client.

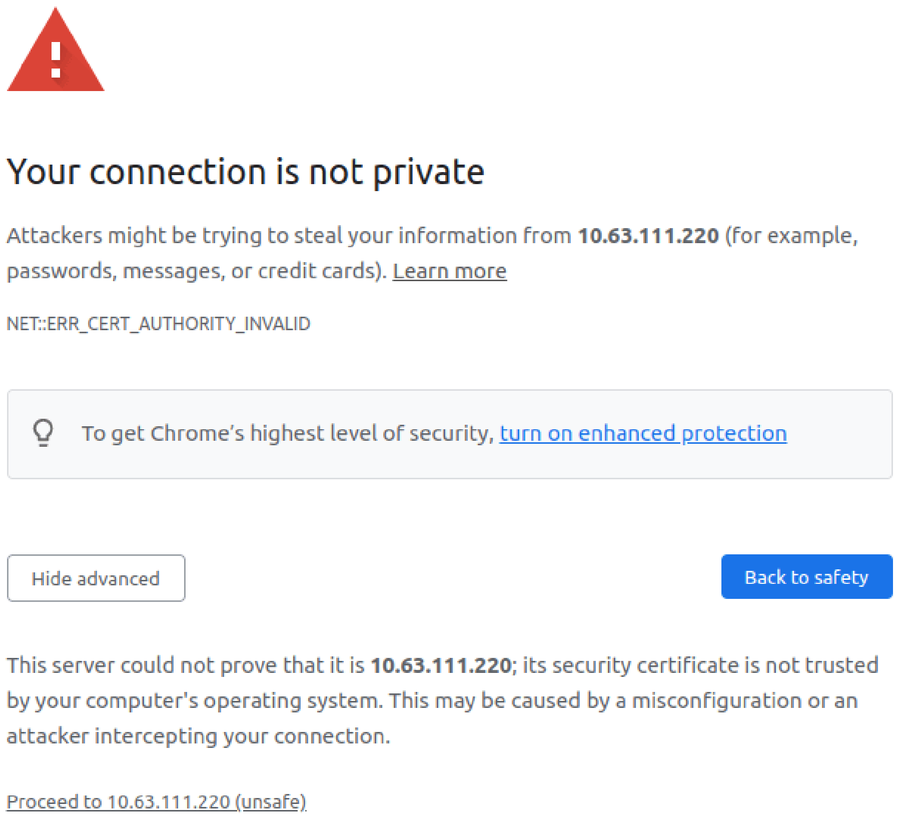
Complete the following steps to access the LXD web UI:

1. Make sure that your LXD server is *exposed to the network* (page 44). You can expose the server during *initialization* (page 35), or afterwards by setting the *core.https_address* (page 402) server configuration option.

[43] https://github.com/canonical/lxd-ui

2. Access the UI in your browser by entering the server address (for example, `https://127.0.0.1:8443` for a local server, or an address like `https://192.0.2.10:8443` for a server running on `192.0.2.10`).

   If you have not set up a secure *TLS server certificate* (page 361), LXD uses a self-signed certificate, which will cause a security warning in your browser. Use your browser's mechanism to continue despite the security warning.

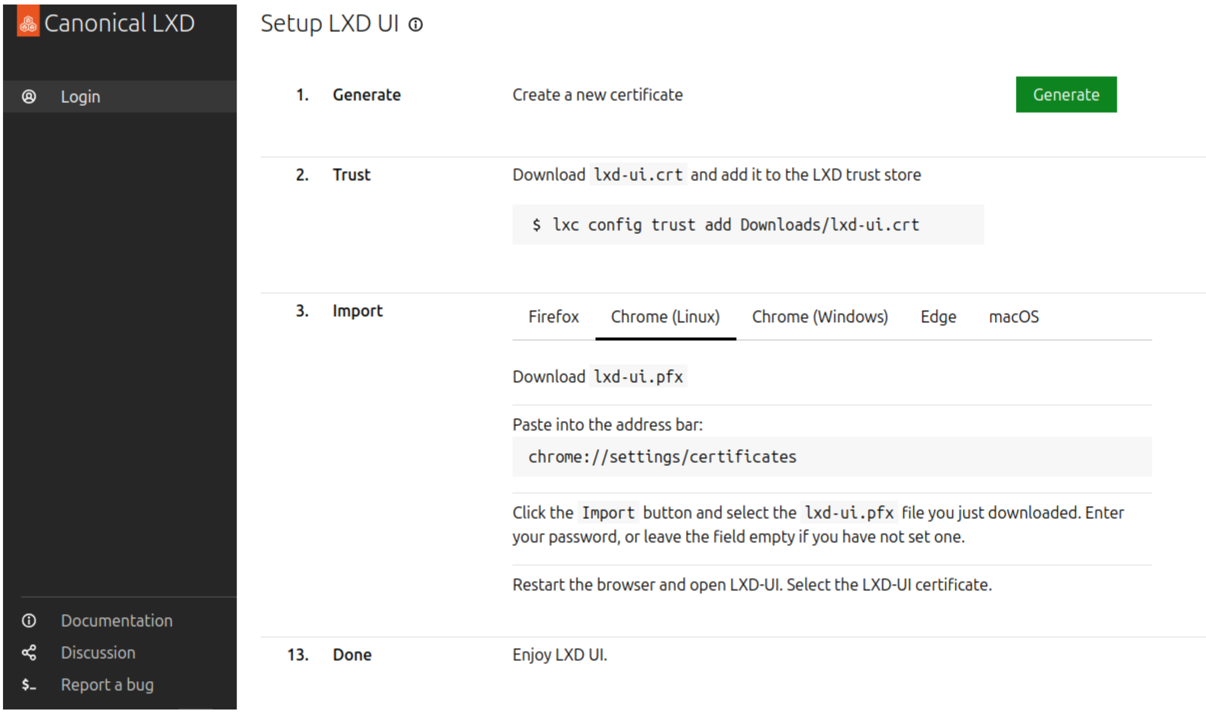
3. Set up the certificates that are required for the UI client to authenticate with the LXD server by following the steps presented in the UI.

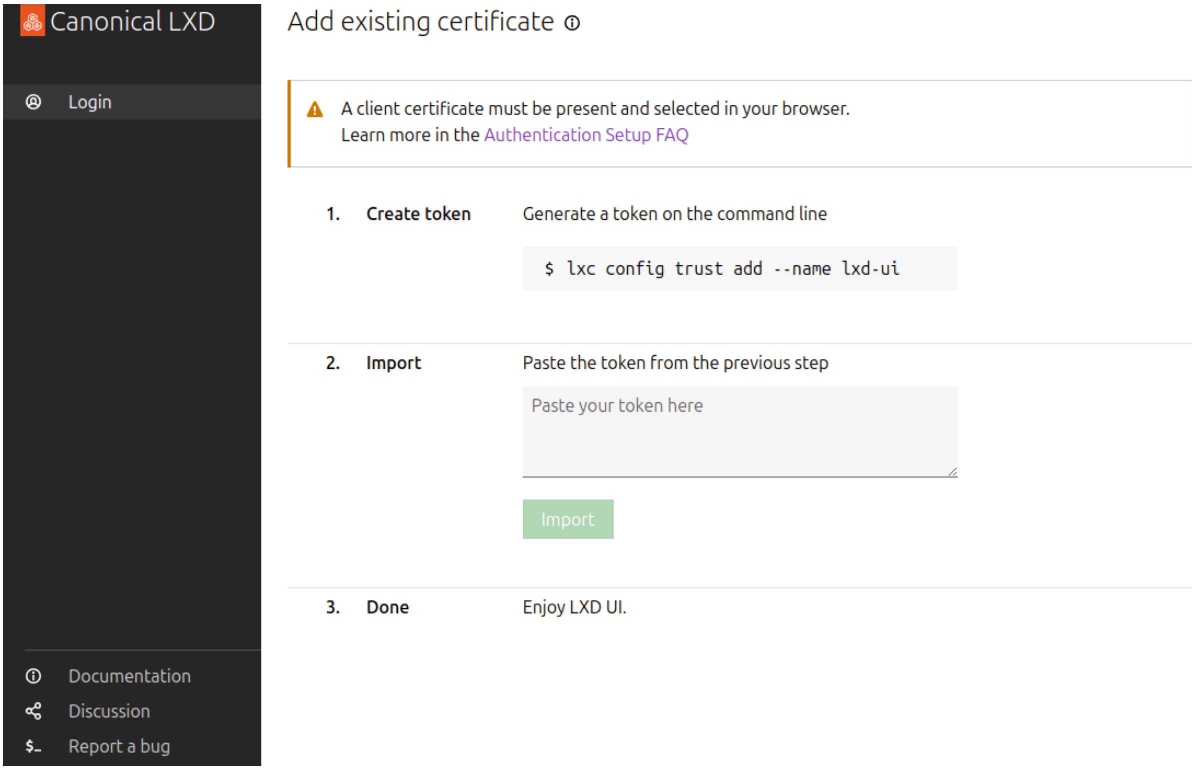
   You have two options, depending on whether you already have a client certificate selected in your browser:

   - If you don't have a certificate yet, click *Create a new certificate* to get instructions for creating a set of certificates, adding the public key to the server's trust store, and adding the private key to your browser.
   - If you already have a client certificate in your browser, select "use an existing certificate" to authorize the certificate with the server and re-use it.

   See *Remote API authentication* (page 358) for more information.

After setting up the certificates, you can start creating instances, editing profiles, or configuring your server.


Canonical LXD
Login
Documentation
Discussion
Report a bug
Setup LXD UI
1. Generate
Create a new certificate
Generate
2. Trust
Download lxd-ui.crt and add it to the LXD trust store
$ lxc config trust add Downloads/lxd-ui.crt
3. Import
Firefox
Chrome (Linux)
Chrome (Windows)
Edge
macOS
Download lxd-ui.pfx
Paste into the address bar:
chrome://settings/certificates
Click the Import button and select the lxd-ui.pfx file you just downloaded. Enter your password, or leave the field empty if you have not set one.
Restart the browser and open LXD-UI. Select the LXD-UI certificate.
13. Done
Enjoy LXD UI.


Canonical LXD
Login
Documentation
Discussion
Report a bug
Add existing certificate
A client certificate must be present and selected in your browser.
Learn more in the Authentication Setup FAQ
1. Create token
Generate a token on the command line
$ lxc config trust add --name lxd-ui
2. Import
Paste the token from the previous step
Paste your token here
Import
3. Done
Enjoy LXD UI.

Enable or disable the UI

The *snap configuration option* (page 324) `lxd ui.enable` controls whether the UI is enabled for LXD.

Starting with LXD 5.21, the UI is enabled by default. If you want to disable it, set the option to `false`:

```
sudo snap set lxd ui.enable=false
sudo systemctl reload snap.lxd.daemon
```

To enable it again, or to enable it for older LXD versions (that include the UI), set the option to `true`:

```
sudo snap set lxd ui.enable=true
sudo systemctl reload snap.lxd.daemon
```

How to access the local LXD documentation

The latest version of the LXD documentation is available at `documentation.ubuntu.com/lxd`[44].

Alternatively, you can access a local version of the LXD documentation that is embedded in the LXD snap. This version of the documentation exactly matches the version of your LXD deployment, but might be missing additions, fixes, or clarifications that were added after the release of the snap.

Complete the following steps to access the local LXD documentation:

1. Make sure that your LXD server is *exposed to the network* (page 44). You can expose the server during *initialization* (page 35), or afterwards by setting the *`core.https_address`* (page 402) server configuration option.
2. Access the documentation in your browser by entering the server address followed by `/documentation/` (for example, `https://192.0.2.10:8443/documentation/`).

   If you have not set up a secure *TLS server certificate* (page 361), LXD uses a self-signed certificate, which will cause a security warning in your browser. Use your browser's mechanism to continue despite the security warning.

In addition, the following clip gives a quick and easy introduction for standard use cases:

You can also find a series of demos and tutorials on YouTube:

Related topics

Tutorials:

- *First steps with LXD* (page 4)
- *Getting started with the UI* (page 11)

Explanation:

- *Containers and VMs* (page 346)

Reference:

- *Requirements* (page 385)

[44] https://documentation.ubuntu.com/lxd/

- *Releases and snap* (page 388)

2.1.2. LXD server and client

The following how-to guides cover common operations related to the LXD server:

How to expose LXD to the network

By default, LXD can be used only by local users through a Unix socket and is not accessible over the network.

To expose LXD to the network, you must configure it to listen to addresses other than the local Unix socket. To do so, set the *core.https_address* (page 402) server configuration option.

For example, allow access to the LXD server on port 8443:

CLI

API

UI

```
lxc config set core.https_address :8443
```

```
lxc query --request PATCH /1.0 --data '{
  "config": {
    "core.https_address": ":8443"
  }
}'
```

Note

The UI requires LXD to be exposed to the network. Therefore, you must use the CLI or API to originally expose LXD to the network.

Once you have access to the UI, you can use it to update the setting. However, be careful when changing the configured value, because using an invalid value might cause you to lose access to the UI.

Go to *Settings* and edit the value for `core.https_address`.

To allow access through a specific IP address, use `ip addr` to find an available address and then set it. For example:

```
~$ ip addr 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN group default qlen 1000 link/loopback 00:00:00:00:00:00 brd
00:00:00:00:00:00 inet 127.0.0.1/8 scope host lo valid_lft forever
preferred_lft forever inet6 ::1/128 scope host valid_lft forever
preferred_lft forever2: enp5s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
qdisc mq state UP group default qlen 1000 link/ether 00:16:3e:e3:f3:3f brd
ff:ff:ff:ff:ff:ff inet 10.68.216.12/24 metric 100 brd 10.68.216.255 scope
global dynamic enp5s0 valid_lft 3028sec preferred_lft 3028sec inet6
fd42:e819:7a51:5a7b:216:3eff:fee3:f33f/64 scope global mngtmpaddr
noprefixroute valid_lft forever preferred_lft forever inet6
```

```
fe80::216:3eff:fee3:f33f/64 scope link valid_lft forever preferred_lft
forever3: lxdbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue
state DOWN group default qlen 1000 link/ether 00:16:3e:8d:f3:72 brd
ff:ff:ff:ff:ff:ff inet 10.64.82.1/24 scope global lxdbr0 valid_lft forever
preferred_lft forever inet6 fd42:f4ab:4399:e6eb::1/64 scope global valid_lft
forever preferred_lft forever ~$ lxc config set core.https_address
10.68.216.12
```

All remote clients can then connect to LXD and access any image that is marked for public use.

Authenticate with the LXD server

To be able to access the remote API, clients must authenticate with the LXD server. There are several authentication methods; see *Remote API authentication* (page 358) for detailed information.

The recommended method is to add the client's TLS certificate to the server's trust store through a trust token. There are two ways to create a token. Create a *pending fine-grained TLS identity* if you would like to manage client permissions via *Fine-grained authorization* (page 364). Create a *certificate add token* if you would like to grant the client full access to LXD, or manage their permissions via *Restricted TLS certificates* (page 364).

See *How to access the LXD web UI* (page 40) for instructions on how to authenticate with the LXD server using the UI. To authenticate a CLI or API client using a trust token, complete the following steps:

1. On the server, generate a trust token.

   CLI

   API

   There are currently two ways to retrieve a trust token in LXD.

   Create a certificate add token

   To generate a trust token, enter the following command on the server:

   ```
   lxc config trust add
   ```

   Enter the name of the client that you want to add. The command generates and prints a token that can be used to add the client certificate.

   > **Note**
   >
   > The recipient of this token will have full access to LXD. To restrict the access of the client, you must use the `--restricted` flag. See *Confine users to specific projects on the HTTPS API* (page 169) for more details.

   Create a pending fine-grained TLS identity

   To create a pending fine-grained TLS identity, enter the following command on the server:

```
lxc auth identity create tls/<client_name>
```

The command generates and prints a token that can be used to add the client certificate.

Note

The recipient of this token is not authorized to perform any actions in the LXD server. To grant access, the identity must be added to one or more groups with permissions assigned. See *Fine-grained authorization* (page 364).

Create a certificate add token

To generate a trust token, send a POST request to the `/1.0/certificates` endpoint:

```
lxc query --request POST /1.0/certificates --data '{
  "name": "<client_name>",
  "token": true,
  "type": "client"
}'
```

See `POST /1.0/certificates` for more information.

The return value of this query contains an operation that has the information that is required to generate the trust token:

```
{
 "class": "token",
 ...
 "metadata": {
    "addresses": [
       "<server_address>"
    ],
    "fingerprint": "<fingerprint>",
    ...
    "secret": "<secret>"
 },
 ...
}
```

Use this information to generate the trust token:

```
echo -n '{"client_name":"<client_name>","fingerprint":"<fingerprint>",'\
'"addresses":["<server_address>"],'\
'"secret":"<secret>","expires_at":"0001-01-01T00:00:00Z"}' | base64 -w0
```

Create a pending fine-grained TLS identity

To generate a trust token, send a POST request to the `/1.0/auth/identities/tls` endpoint:

```
lxc query --request POST /1.0/auth/identities/tls --data '{
  "name": "<client_name>",
  "token": true
}'
```

See `POST /1.0/auth/identities/tls` for more information.

The return value of this query contains the information that is required to generate the trust token:

```
{
    "client_name": "<client_name>",
    "addresses": [
       "<server_address>"
    ],
    "expires_at": "<expiry_date>"
    "fingerprint": "<fingerprint>",
    "type": "<type>",
    "secret": "<secret>"
}
```

Use this information to generate the trust token:

```
echo -n '{"client_name":"<client_name>","fingerprint":"<fingerprint>",'\
'"addresses":["<server_address>"],'\
'"secret":"<secret>","expires_at":"0001-01-01T00:00:00Z","type":"<type>"}' |
base64 -w0
```

2. Authenticate the client.

   CLI

   API

   On the client, add the server with the following command:

```
lxc remote add <remote_name> <token>
```

> **Note**
>
> If your LXD server is behind NAT, you must specify its external public address when adding it as a remote for a client:
>
> ```
> lxc remote add <name> <IP_address>
> ```
>
> When you are prompted for the token, specify the generated token from the previous step. Alternatively, use the `--token` flag:
>
> ```
> lxc remote add <name> <IP_address> --token <token>
> ```
>
> When generating the token on the server, LXD includes a list of IP addresses that the client can use to access the server. However, if the server is behind NAT, these addresses might be local addresses that the client cannot connect to. In this case, you must specify the external address manually.

On the client, generate a certificate to use for the connection:

```
openssl req -x509 -newkey rsa:2048 -keyout "<keyfile_name>" -nodes \
-out "<crtfile_name>" -subj "/CN=<client_name>"
```

Trust store entries

Then send a POST request to the `/1.0/certificates?public` endpoint to authenticate:

```
curl -k -s --key "<keyfile_name>" --cert "<crtfile_name>" \
-X POST https://<server_address>/1.0/certificates \
--data '{ "trust_token": "<trust_token>" }'
```

See `POST /1.0/certificates?public` for more information.

TLS identities

Send a POST request to the `/1.0/auth/identities/tls?public` endpoint to authenticate:

```
curl --insecure --key "<keyfile_name>" --cert "<crtfile_name>" \
-X POST https://<server_address>/1.0/auth/identities/tls \
--data '{ "trust_token": "<trust_token>" }'
```

See `POST /1.0/auth/identities/tls?public` for more information.

See *Remote API authentication* (page 358) for detailed information and other authentication methods.

How to configure the LXD server

See *Server configuration* (page 401) for all configuration options that are available for the LXD server.

If the LXD server is part of a cluster, some of the options apply to the cluster, while others apply only to the local server, thus the cluster member. In the *Server configuration* (page 401) option tables, options that apply to the cluster are marked with a `global` scope, while options that apply to the local server are marked with a `local` scope.

Configure server options

CLI

API

UI

You can configure a server option with the following command:

```
lxc config set <key> <value>
```

For example, to allow remote access to the LXD server on port 8443, enter the following command:

```
lxc config set core.https_address :8443
```

In a cluster setup, to configure a server option for a cluster member only, add the `--target` flag. For example, to configure where to store image tarballs on a specific cluster member, enter a command similar to the following:

```
lxc config set storage.images_volume my-pool/my-volume --target member02
```

Send a PATCH request to the `/1.0` endpoint to update one or more server options:

```
lxc query --request PATCH /1.0 --data '{
  "config": {
    "<key>": "<value>",
    "<key>": "<value>"
  }
}'
```

For example, to allow remote access to the LXD server on port 8443, send the following request:

```
lxc query --request PATCH /1.0 --data '{
  "config": {
    "core.https_address": ":8443"
  }
}'
```

In a cluster setup, to configure a server option for a cluster member only, add the `target` parameter to the query. For example, to configure where to store image tarballs on a specific cluster member, send a request similar to the following:

```
lxc query --request PATCH /1.0?target=member02 --data '{
  "config": {
    "storage.images_volume": "my-pool/my-volume"
  }
}'
```

See `PATCH /1.0` for more information.

Go to *Settings* to configure the server options.

In a cluster setup, server options that apply to the local server are updated only for the server on which you are accessing the UI. For example, if you access the UI on `server1` and update the location for storing image tarballs (*`storage.images_volume`* (page 413), which has a `local` scope) from `local/pool1` to `local/pool2`, `storage.images_volume` will still be configured to `local/pool1` on `server2`.

Display the server configuration

CLI

API

UI

To display the current server configuration, enter the following command:

```
lxc config show
```

In a cluster setup, to show the local configuration for a specific cluster member, add the `--target` flag.

Send a GET request to the `/1.0` endpoint to display the current server environment and configuration:

```
lxc query --request GET /1.0
```

In a cluster setup, to show the local environment and configuration for a specific cluster member, add the `target` parameter to the query:

```
lxc query --request GET /1.0?target=<cluster_member>
```

See `GET /1.0` for more information.

Go to *Settings* to view the current server configuration.

In a cluster setup, this view shows the local configuration for the cluster member on which you are accessing the UI.

Edit the full server configuration

CLI

API

UI

To edit the full server configuration as a YAML file, enter the following command:

```
lxc config edit
```

In a cluster setup, to edit the local configuration for a specific cluster member, add the `--target` flag.

To update the full server configuration, send a PUT request to the `/1.0` endpoint:

```
lxc query --request PUT /1.0 --data '<server_configuration>'
```

In a cluster setup, to update the full server configuration for a specific cluster member, add the `target` parameter to the query:

```
lxc query --request PUT /1.0?target=<cluster_member> '<server_configuration>'
```

See `PUT /1.0` for more information.

The UI does not currently support editing the full server configuration.

How to configure Auth0 as login method for the LXD UI and CLI

Auth0 is a flexible, drop-in solution to add authentication and authorization services to your applications. Auth0 supports OIDC and can be used to authenticate users for the LXD UI and CLI. This guide shows you how to set up Auth0.com as the login method for the LXD UI and CLI.

Using Auth0.com to access LXD

1. Open a free account on Auth0.com[45].
2. Once logged into the Auth0 web interface, from the main navigation's *Applications* section, select *Applications* > *Create application*.
   - Use the default type of *Native* and click *Create*.
3. Go to the *Settings* tab of your new application.
   - Scroll to the *Allowed Callback URLs* field in this tab and enter your LXD UI address, followed by `/oidc/callback`.
     - Example: `https://example.com:8443/oidc/callback`
     - An IP address can be used instead of a domain name.
     - Note `:8443` is the default listening port for the LXD server. It might differ for your setup. You can verify the LXD configuration value `core.https_address` to find the correct port for your LXD server.
   - Enable *Allow Refresh Token Rotation*.
   - Scroll down to *Advanced Settings* and select the *Grant Types* tab. Enable *Device code*. The device code grant type is required for OIDC authentication using the LXD CLI.
   - Select *Save Changes*.
4. Near the top of the *Settings* tab, locate the *Domain* field. Copy the value and add the `https://` prefix and the `/` suffix as in the example below. This is your OIDC issuer for LXD. Set this value in your LXD server configuration with the command:

   ```
   lxc config set oidc.issuer=https://dev-example.us.auth0.com/
   ```

5. From your Auth0 application's *Settings* tab, copy the *Client ID* and use it in your LXD server configuration:

   ```
   lxc config set oidc.client.id=6f6f6f6f6f6f
   ```

6. Finally, in the Auth0 interface's main navigation, under *Applications*, select *Applications* > *APIs*. Copy the *API Audience* value, then use it as the OIDC audience in your LXD server configuration:

   ```
   lxc config set oidc.audience=https://dev-example.us.auth0.com/api/v2/
   ```

Now you can access the LXD UI with any browser and use SSO (single sign-on) login. Enter the credentials for Auth0. You can also access LXD using the CLI with

```
lxc remote add <remote-name> <LXD-address> --auth-type oidc
```

This will open a browser where you must confirm the device code displayed in the terminal window, and log in with the credentials for Auth0.

[45] https://auth0.com/

Users will have no permissions by default. You must set up *LXD authorization groups* (page 366) to grant access to projects and instances. For connecting the LXD authorization groups to a user you have two options:

1. Map a LXD authorization group to the user directly. Note, that the user object in LXD will only be created on the first login of that user to LXD.
2. Configure roles in Auth0 and use automatic mapping to LXD authorization groups as described below.

Set up automatic group mappings

An admin can set up multiple users in Auth0 and allocate roles to those users. When a user logs in via OIDC, their allocated Auth0 roles can be mapped to LXD authorization groups through custom claims. This section details the steps for configuring roles in Auth0 and setting up a custom claim so that LXD can map those roles to its authorization groups.

1. In Auth0 interface for the application used with LXD, select *User Management* > *Roles*, create some roles with suitable names.
2. Under *User Management* > *Users*, click *Create User*. Provide an email and password and create the user.
3. Select on the *Roles* tab in the user detail page, then click the *Assign Roles* button. Select the roles you created in step 1.
4. You must set up a custom action on Auth0 to set the custom claim on the id_token during the OIDC login flow.
   - In the main navigation, under *Actions* > *Library*, click the *Create Action* button. Select *Build from scratch*.
     - **Name**: Give the action a suitable name like `roles-in-id-token`.
     - **Trigger**: Login / Post Login
     - **Runtime**: The recommended default
   - Click *Create*. This causes a code editor to open.
   - In the code editor, insert the code snippet shown below:

```
exports.onExecutePostLogin = async (event, api) => {
  if (event.authorization) {
    api.idToken.setCustomClaim(`lxd-idp-groups`, event.authorization.roles);
    api.accessToken.setCustomClaim(`lxd-idp-groups`, event.authorization.
roles);
  }
};
```

   - Select *Deploy*.
   - Once the action is deployed, go to *Actions* > *Triggers* > *post-login*. Under the *Add Action* > *Custom* tab, drag the action you just created and drop it in between the *Start* and *Complete* nodes of the Login flow. Select *Apply* to save the changes.
5. Navigate to the LXD UI. First authenticate with the UI using a trusted certificate so that you can configure server settings without permission issues.

6. In the LXD UI, under *settings*, find `oidc.groups.claim`. Set it to the custom claim configured in step 4. Using the current example, the custom claim is `lxd-idp-groups`. Alternatively, use the command line: `lxc config set oidc.groups.claim=lxd-idp-groups`.
7. Continuing in the LXD UI, navigate to *Permissions* > *IDP groups* and click *Create IDP Group*. Here you can map roles from Auth0 to LXD authorization groups. For each *identity provider group* (page 367) created in LXD, the name of the identity provider group must match a role you have created in Auth0, and it should also map to one or more LXD authorization groups. Alternatively, use the command line:

```
lxc auth identity-provider-group create <auth0-role-name>
lxc auth identity-provider-group group add <auth0-role-name> <LXD-group-name>
```

8. Lastly, you log in as a user with roles assigned in Auth0. During the OIDC flow, LXD automatically extracts the custom claim from the user's `id_token` based on the LXD `oidc.groups.claim` configuration value. The extracted custom claim is an array of roles for your user from Auth0. Those roles are then mapped to LXD authorization groups using the identity provider group created in step 7.

How to configure Ory Hydra as login method for the LXD UI

Ory Hydra is an easy solution to authenticate users for the LXD UI. It supports local users and social sign in through Google, Facebook, Microsoft, GitHub, Apple or others. It does not yet work for the LXD command line. This guide shows you how to set up Ory Hydra as the login method for the LXD UI.

Using Ory Hydra to access LXD UI

1. Open a free account on Ory.sh/Hydra[46].
2. Once logged into the Ory Console, navigate to *OAuth 2* > *OAuth2 Clients* > *Create OAuth2 Client*.
3. Select the type *Mobile / SPA* and click *Create*. Enter the details for the client:
   - **Client Name**: Choose a name, such as `lxd-ory-client`.
   - **Scope**: Enter `email` and click *Add*, then add `profile` as well.
   - **Redirect URIs**: Enter your LXD UI address, followed by `/oidc/callback`, then click *Add*.
     - Example: `https://example.com:8443/oidc/callback`
     - An IP address can be used instead of a domain name.
     - Note: `:8443` is the default listening port for the LXD server. It might differ for your setup. Use `lxc config get core.https_address` to find the correct port for your LXD server.
4. Select *Create Client* on the bottom of the page.
5. On the *OAuth2 Clients* list, find the *ID* for the client you created. Copy the value and set it in your LXD server configuration with the command:

[46] https://www.ory.sh/hydra/

```
lxc config set oidc.client.id=<your OAuth2 Client ID>
```

6. In the Ory Console, navigate to *OAuth 2* > *Overview*. Find the *Issuer URL* and copy the value. Set this value in your LXD server configuration as issuer with the commands:

```
lxc config set oidc.issuer=https://<ory-id>.projects.oryapis.com
```

Now you can access the LXD UI with any browser and use SSO login.

No users exist within ORY by default. New users can use the sign-up link during login. Alternatively, configure Google, Facebook, Microsoft, GitHub, Apple, or another social sign-in provider as described in the ORY documentation[47].

Users authenticated through ORY have no default permissions in the LXD UI. Set up *LXD authorization groups* (page 366) to grant access to projects and instances and map a LXD authorization group to the user. Note that the user object in LXD is only created on the first login of that user to LXD.

How to configure Keycloak as login method for LXD

Keycloak is a self-hosted open source tool for authentication. Keycloak supports OIDC and can be used to authenticate users for LXD UI and CLI. This guide shows you how to set up Keycloak as the login method for LXD.

Using Keycloak to access LXD

1. Set up Keycloak. For this guide, it is assumed that Keycloak is available over HTTPS.
   - If you already have Keycloak installed, follow their guide on configuring Keycloak for production[48].
   - Alternatively, run the development version:
     - Download Keycloak-25.0.4[49].
     - Extract the files and run `bin/kc.sh start-dev`.
     - Open `http://localhost:8080` in your browser and create an admin user with a password.
2. Open the Keycloak Admin Console. For the development version, you can access this at `http://localhost:8080/admin`. Sign in with the admin user that you created.
3. From the *Keycloak* dropdown in the top left corner of the Admin Console, select *Create realm*. Enter a *Realm name*, such as `lxd-ui-realm`, then click *Create*.
4. From the main navigation, select *Clients*, then click *Create client*. Enter a *Client ID*, such as `lxd-ui-client`, then click *Next*.
5. Enable the *OAuth 2.0 Device Authorization Grant* to allow the LXD CLI login. Click *Next*.
6. In the field for *Valid redirect URIs*, enter your LXD UI address, followed by `/oidc/callback`.
   - Example: `https://example.com:8443/oidc/callback`

[47] https://www.ory.sh/docs/kratos/social-signin/overview
[48] https://www.keycloak.org/server/configuration-production
[49] https://github.com/keycloak/keycloak/releases/download/25.0.4/keycloak-25.0.4.zip

- An IP address can be used instead of a domain name.
- Note :8443 is the default listening port for the LXD server. It might differ for your setup. You can verify the LXD configuration value core.https_address to find the correct port for your LXD server.

Click *Save*.

7. From the main navigation, select *Users*, then click *Create new user*. Enter a *Username*, then click *Create*.
8. Select the *Credentials* tab for the new user and click *Set password*. Save the new password.
9. Configure the issuer on your LXD server via the CLI. For <keycloak-realm>, use the name that you created in step 2. For the <keycloak-frontend-url>, use the URL for your Keycloak server, such as http://192.0.2.1:8080. If you are running the development version of Keycloak, use http://localhost:8080.

```
lxc config set oidc.issuer=<keycloak-frontend-url>/realms/<keycloak-realm>
```

10. Configure the client in LXD with the command below. Use the client id from step 4.

```
lxc config set oidc.client.id=<keycloak-client-id>.
```

Now you can access the LXD UI with any browser and use SSO login. To use OIDC on the LXD CLI, run lxc remote add <LXD address> --auth-type oidc and follow the instructions to authenticate.

Users authenticated through Keycloak have no default permissions in the LXD UI. Set up *LXD authorization groups* (page 366) to grant access to projects and instances and map a LXD authorization group to the user. Note that the user object in LXD is only created on the first login of that user to LXD.

How to configure authentication with Entra ID

Entra ID[50] is an Identity and Access Management offering from Microsoft. It is commonly used as a central location for managing users, groups, roles, and their privileges across many applications and deployments.

LXD supports authentication via OpenID Connect (OIDC)[51] (see *OpenID Connect authentication* (page 361)). Entra ID is an OIDC provider; however, some aspects of the Entra ID OIDC service are non-standard. In particular, the access_token that is returned when a user successfully authenticates using the device authorization grant[52] flow is an opaque string[53], and not a JSON Web Token (JWT)[54].

The LXD CLI uses the device authorization grant flow to obtain an access token. When a command is issued, the CLI adds this token to all requests to the LXD API. For Entra ID, since the token is opaque, LXD is unable to verify it and the command will fail. Therefore, authentication with Entra ID is only directly supported for the user interface (LXD UI) and not the CLI.

[50] https://www.microsoft.com/en-gb/security/business/identity-access/microsoft-entra-id
[51] https://openid.net/
[52] https://tools.ietf.org/html/rfc8628
[53] https://learn.microsoft.com/en-us/entra/identity-platform/v2-oauth2-device-code#successful-authentication-response
[54] https://datatracker.ietf.org/doc/html/rfc7519

We are working toward full Entra ID support for LXD. In the meantime, it is possible to use Entra ID if OIDC is only required for the LXD UI. Alternatively, it is possible to use Entra ID for both the CLI and the user interface by deploying an identity broker such as Keycloak[55].

This how-to guide covers configuring *Entra ID for authentication in the LXD UI only* (page 56), and cover configuring *Keycloak to act as a broker for Entra ID* (page 61). In both cases, it is assumed that LXD has been initialized and is available remotely via the HTTPS API on port 8443 (see *How to expose LXD to the network* (page 44) for instructions). It is also assumed that you have access to an Entra ID tenant.

Using Entra ID directly (LXD UI only)

1. In your Entra ID tenant, go to `Identity > Applications > App registrations` in the left panel.
2. Click `+ New registration`. Then choose a name for the application (for example `LXD`).
3. Under `Redirect URI (optional)`, select `Public client/native (mobile & desktop)` and type:

   ```
   https://<your-LXD-hostname>/oidc/callback
   ```

4. Click `Register`.
5. In the configuration page for your new application, go to `Authentication` in the `Manage` menu.
6. Scroll down to `Advanced settings`. Under `Allow public client flows`, toggle `Yes` and click `Save`.
7. In the configuration page for your new application, go to `API permissions` in the `Manage` menu.
8. Go to `Configured permissions` and click `+ Add a permission`.
9. Click `Microsoft Graph` in the right panel.
10. Click `Delegated permissions`.
11. Select all `OpenId permissions`, then click `Add permissions`.
12. Above the `Manage` menu, go to `Overview` and copy the `Application (client) ID`.
13. Set this as the client ID in LXD:

    ```
    lxc config set oidc.client.id <your-client-id>
    ```

14. While still in `Overview`, click `Endpoints` and copy the URL under `OpenID Connect metadata document`.
15. Navigate to the URL that you copied. This URL will display some output in JSON format.
16. Copy the URL from the `issuer` field. Then set this as the `oidc.issuer` in LXD:

    ```
    lxc config set oidc.issuer <your-issuer>
    ```

    Alternatively, execute this command:

[55] https://www.keycloak.org/

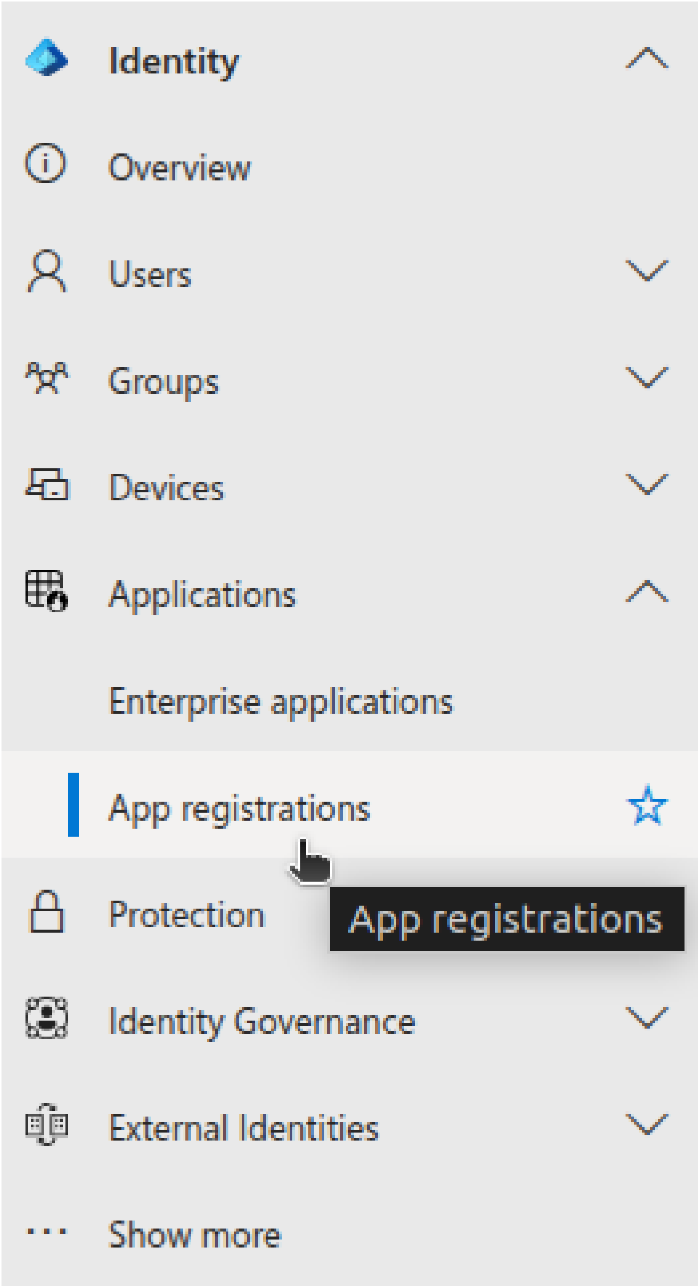


Fig. 2: Entra ID App registrations

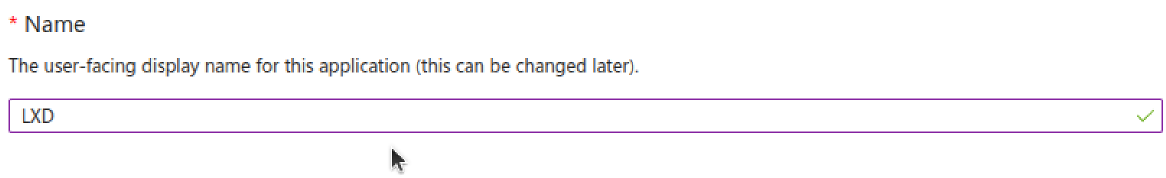


Fig. 3: Entra ID set application name

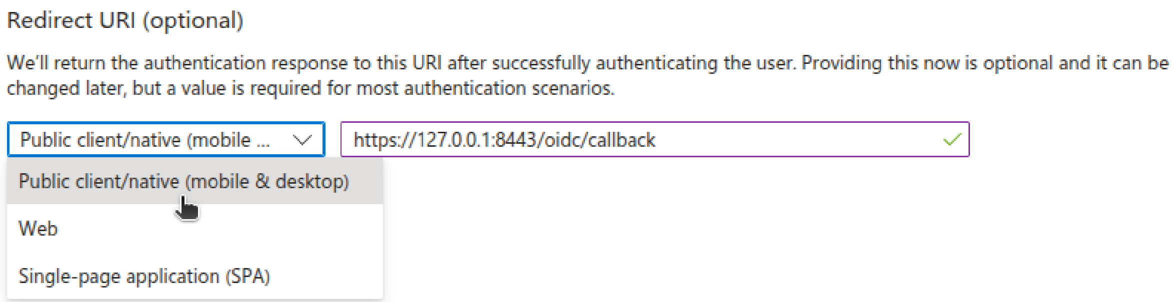


Fig. 4: Entra ID set redirection URI

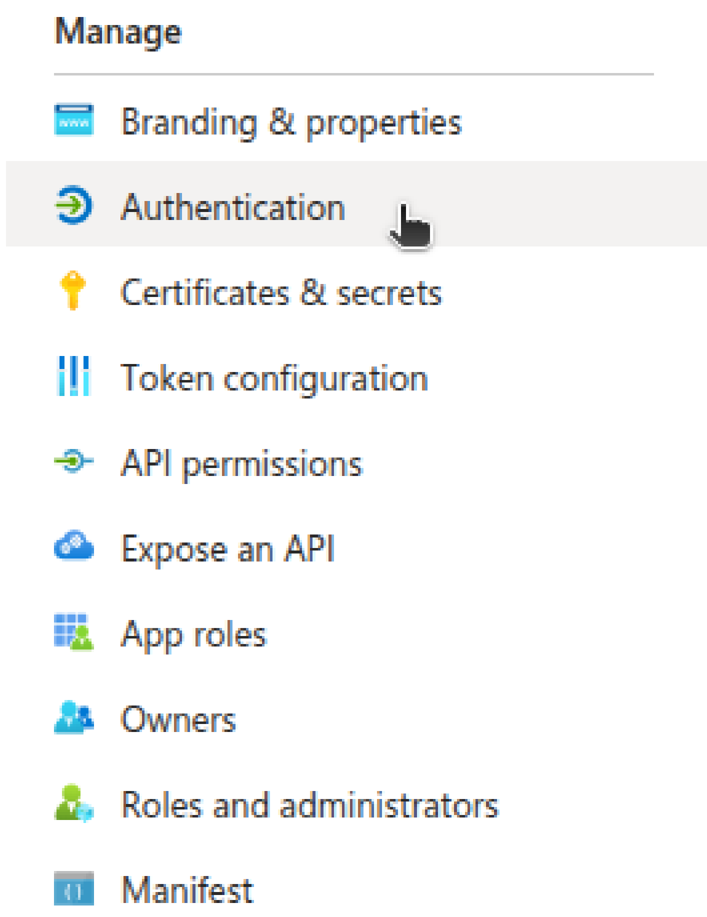


Fig. 5: Entra ID authentication

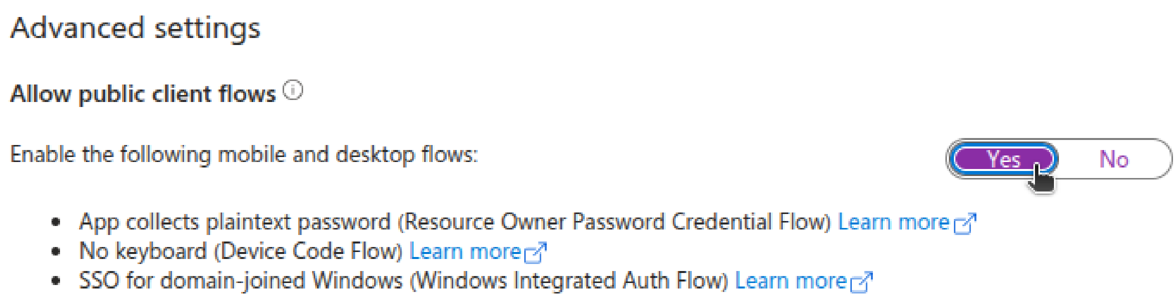


Fig. 6: Entra ID enable public client flows

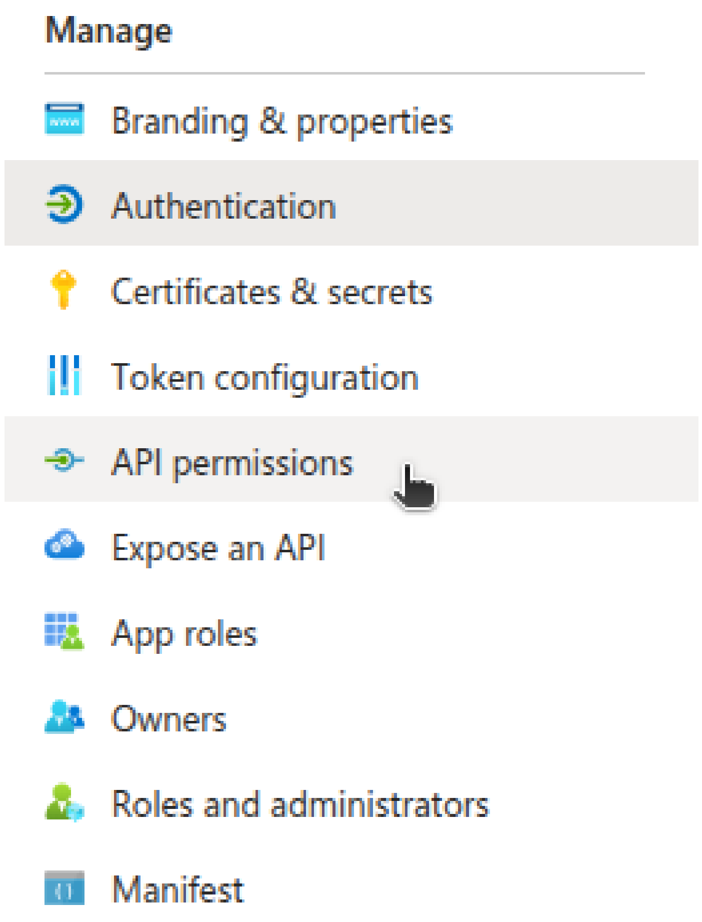


Fig. 7: Entra ID API permissions

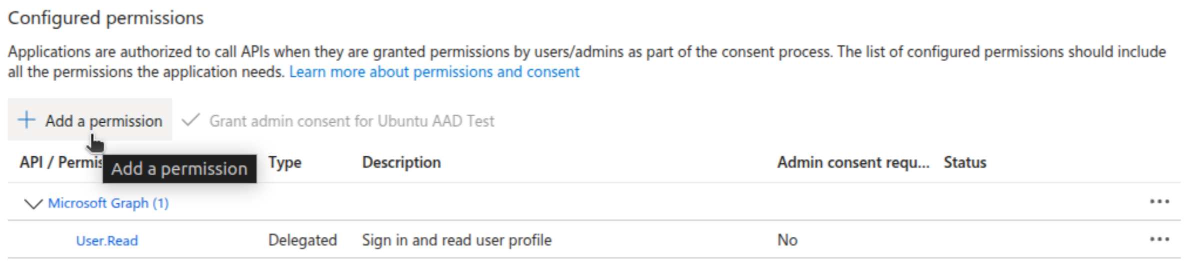


Fig. 8: Entra ID add a permission

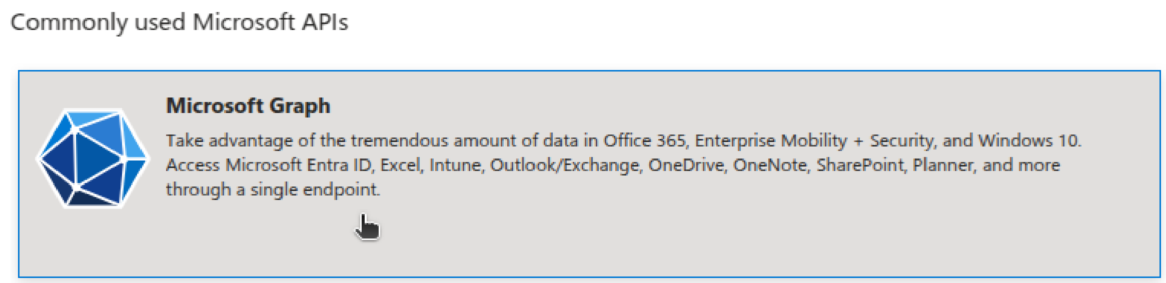


Fig. 9: Entra ID Graph API permissions

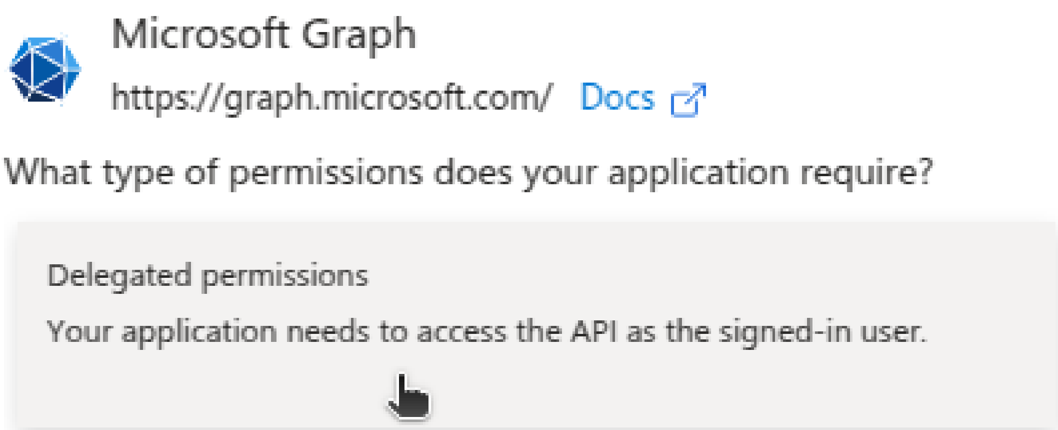


Fig. 10: Entra ID Graph API delegated permissions

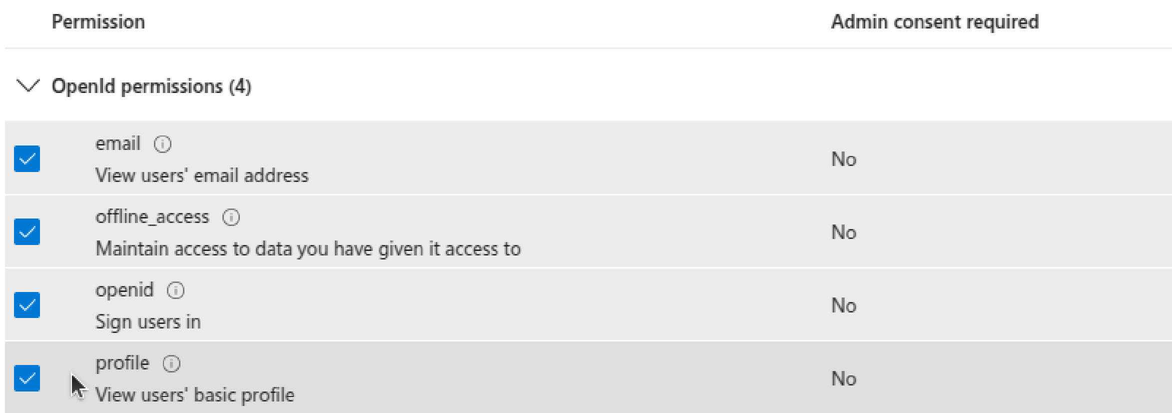


Fig. 11: Entra ID OpenID permissions

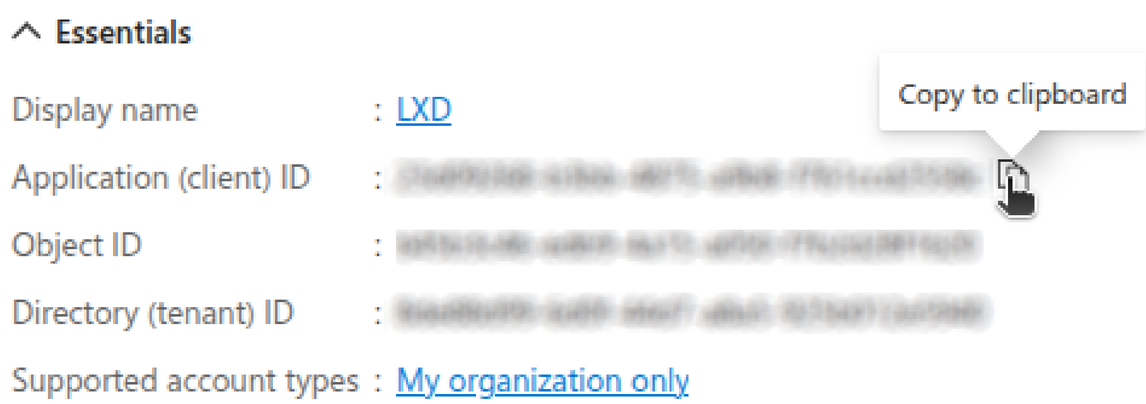


Fig. 12: Entra ID copy client ID

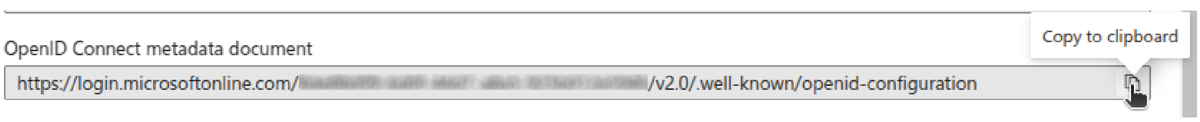


Fig. 13: Entra ID tenant discovery URL

```
lxc config set oidc.issuer "$(curl <URL that you copied> | jq -r .issuer)"
```

You can now navigate to the LXD UI in your browser. When you click `Login with SSO`, you will be redirected to Entra ID to authenticate.

Using Keycloak as an Identity Broker for Entra ID

If you plan to use Keycloak as an identity provider for your production systems, you should follow their guide on configuring Keycloak for production[56]. For this guide, it is assumed that Keycloak is available over HTTPS and that you have created a Keycloak realm with default settings.

1. In your Keycloak realm, go to `Identity providers`.

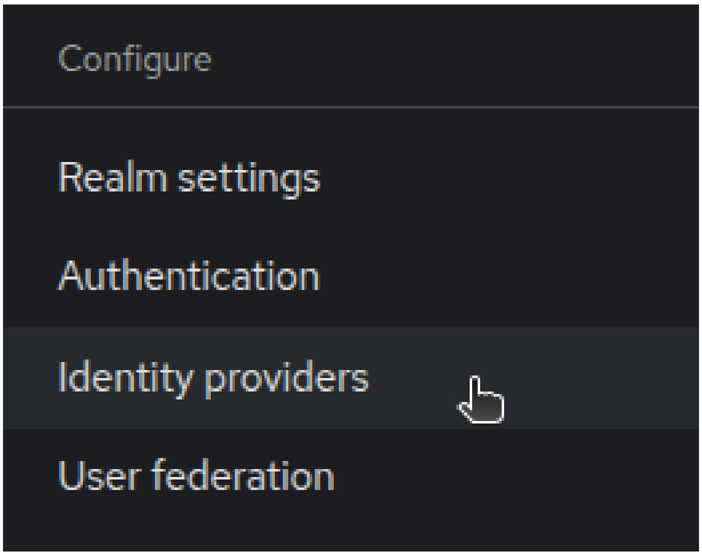


Fig. 14: Keycloak realm Identity providers

2. Click `Microsoft`.

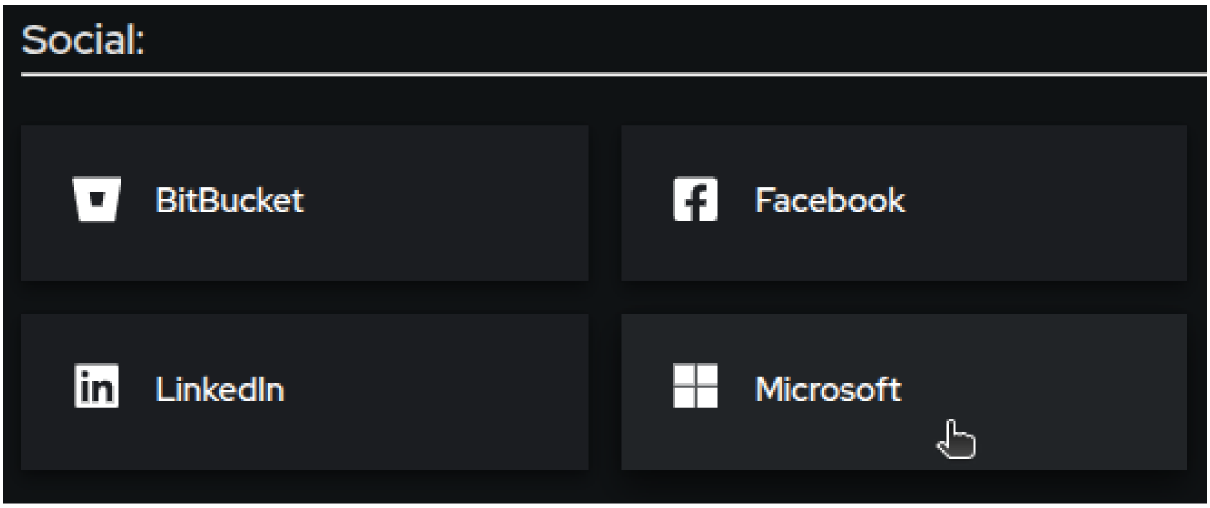


Fig. 15: Keycloak Microsoft provider

[56] https://www.keycloak.org/server/configuration-production

3. On this page, copy the `Redirect URI`. Keep the tab open so that you can return to this page to continue setting up Keycloak.

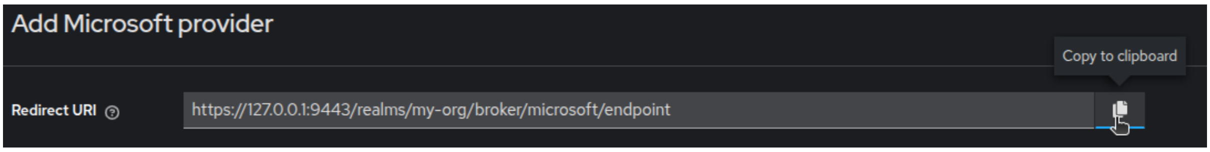


Fig. 16: Keycloak broker redirect URI

4. In your Entra ID tenant, go to `Identity > Applications > App registrations` in the left panel.
5. Click `+ New registration`. Then choose a name for the application (for example `Keycloak`).
6. Under `Redirect URI (optional)`, select `Web` and paste the URL that you copied from Keycloak. Then click `Register`.
7. Go to `Certificates & secrets` under `Manage` in your Entra ID tenant.
8. Click `+ New client secret`.
9. In the right panel, click `Add`. A new client secret will be displayed. Copy the value.

> **Note**
>
> After navigating away from this page, you will no longer be able to view or copy the secret. If you forget to copy it, you can delete it and create another one.

10. In the Keycloak identity provider configuration tab, paste the secret into the `Client secret` field.
11. In Entra ID, go to the app `Overview` and copy the `Application (client) ID`.
12. Paste the value into the `Client ID` field in the Keycloak tab.
13. In Entra ID, go to the app `Overview` and copy the `Directory (tenant) ID`.
14. Paste the value into the `Tenant ID` field in the Keycloak tab.
15. Click `Add`.
16. Follow steps 7 to 11 in the *above guide* (page 56). This allows Keycloak to request the required OpenID scopes.

    We have now configured Keycloak to act as a broker for Entra ID. The remaining steps configure Keycloak as the OIDC provider for LXD.

17. In your Keycloak realm, go to `Clients`.
18. Click `Create client`.
19. Set a `Client ID` and a name for the client, then click `Next`. The client ID in this example is a random value. You can type any value, but it must be unique within the Keycloak realm.
20. In `Authentication flow`, check the `OAuth 2.0 Device Authorization Grant` setting, then click `Next`.

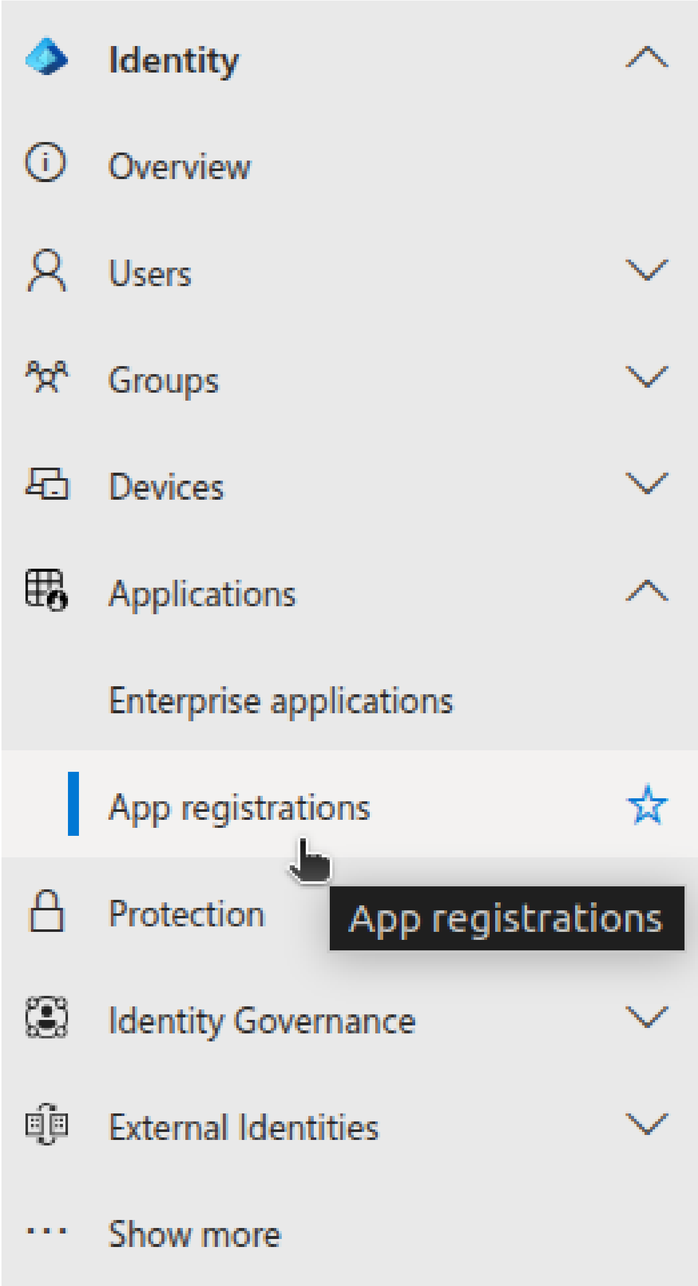


Fig. 17: Entra ID App registrations

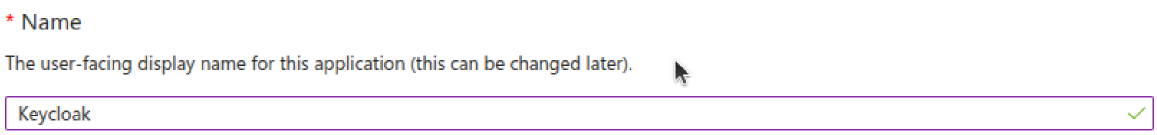


Fig. 18: Entra ID App name

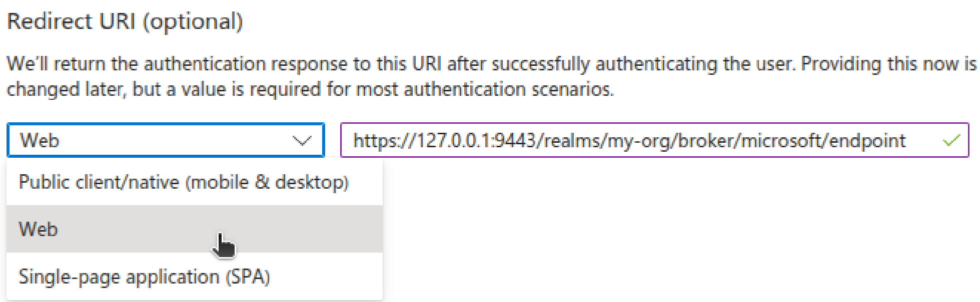


Fig. 19: Entra ID set redirection URI

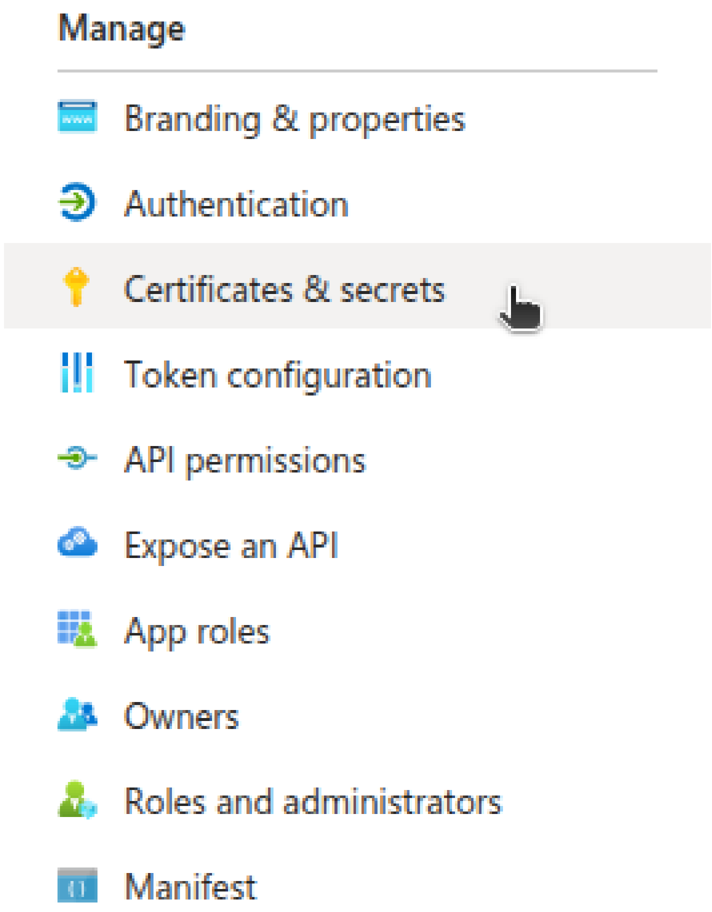


Fig. 20: Entra ID certificates and secrets

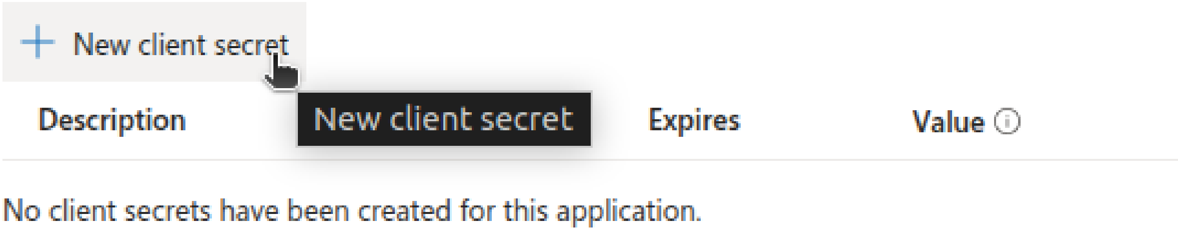


Fig. 21: Entra ID client secret

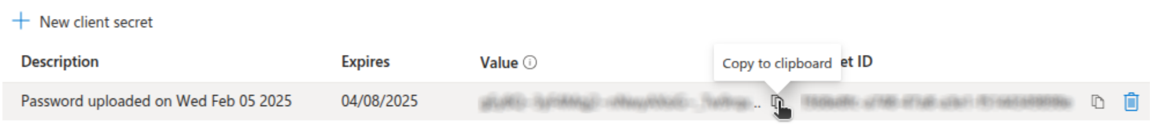


Fig. 22: Entra ID copy client secret

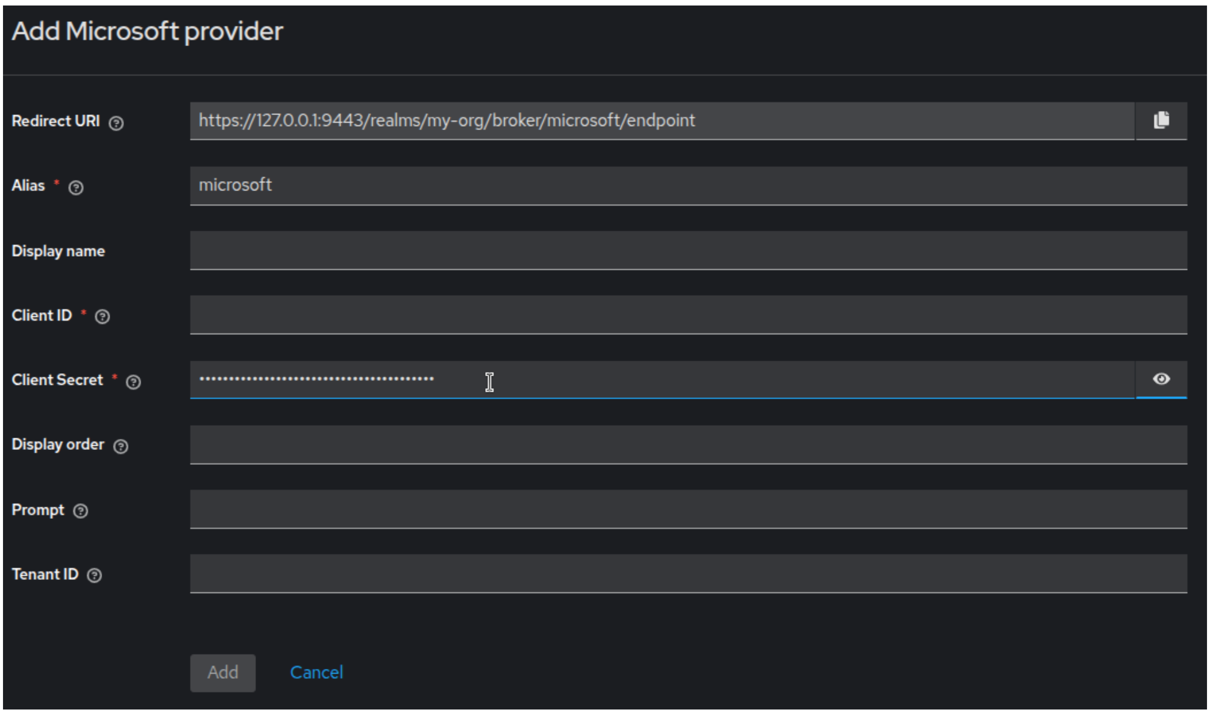


Fig. 23: Keycloak paste client secret

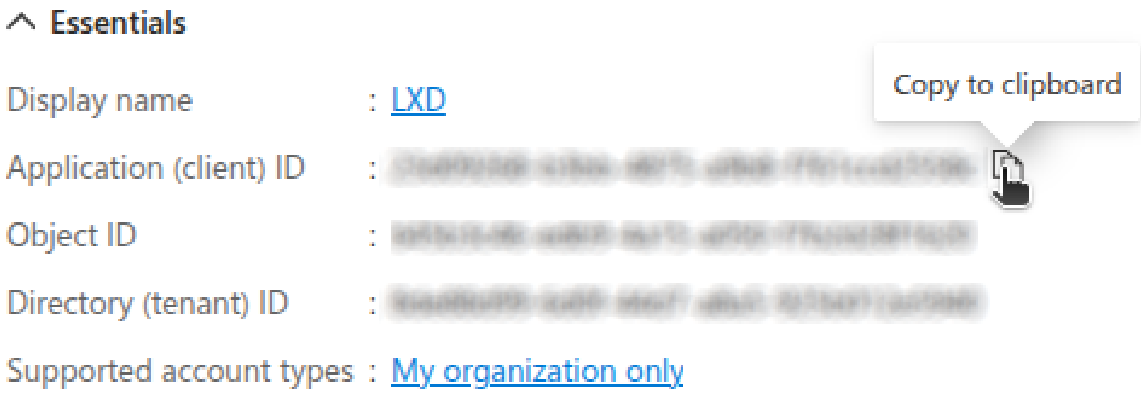


Fig. 24: Entra ID copy client ID

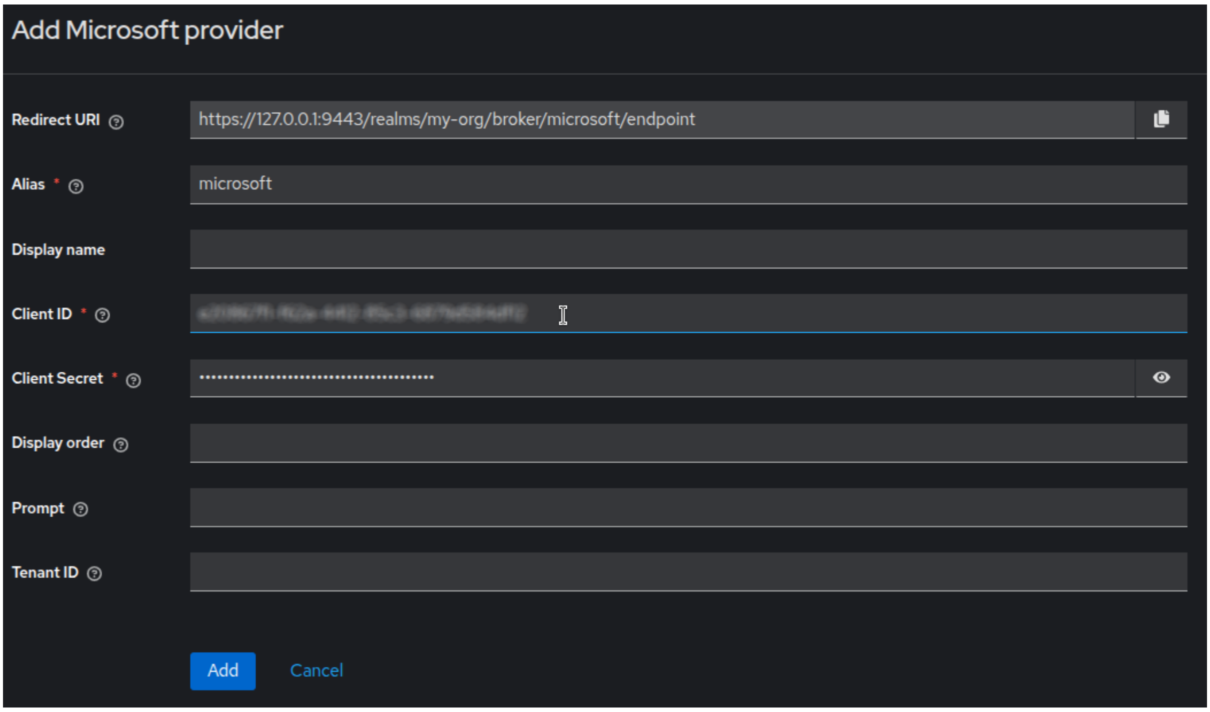


Fig. 25: Keycloak paste client ID

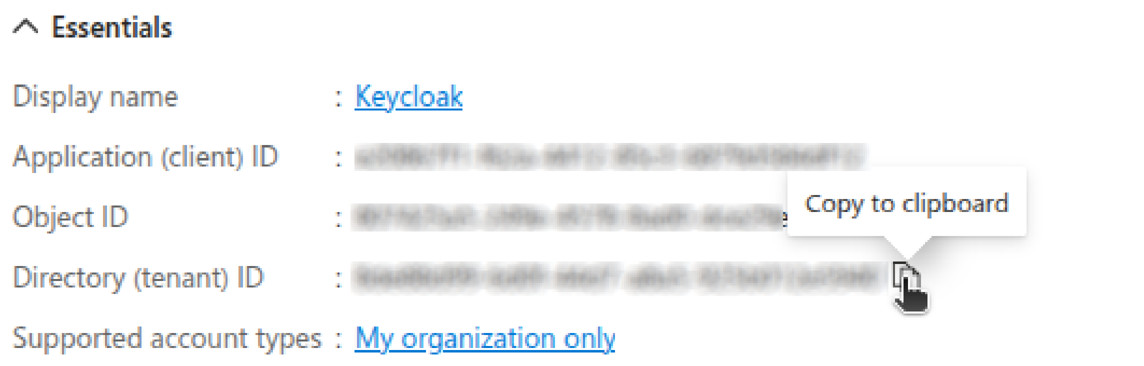


Fig. 26: Entra ID copy tenant ID

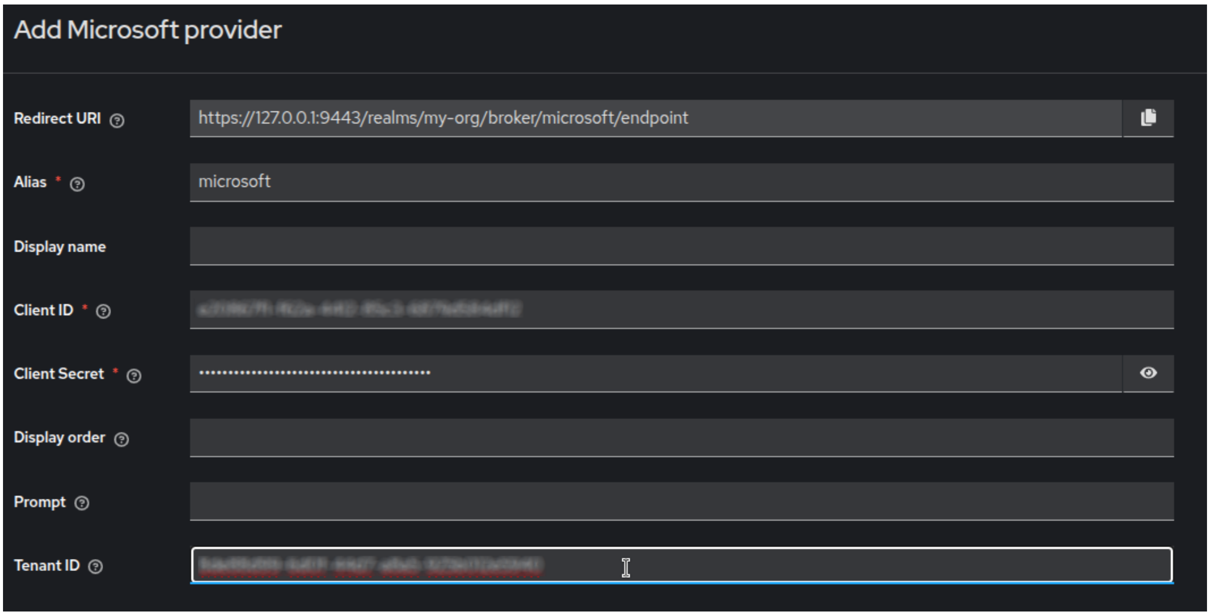


Fig. 27: Keycloak paste tenant ID

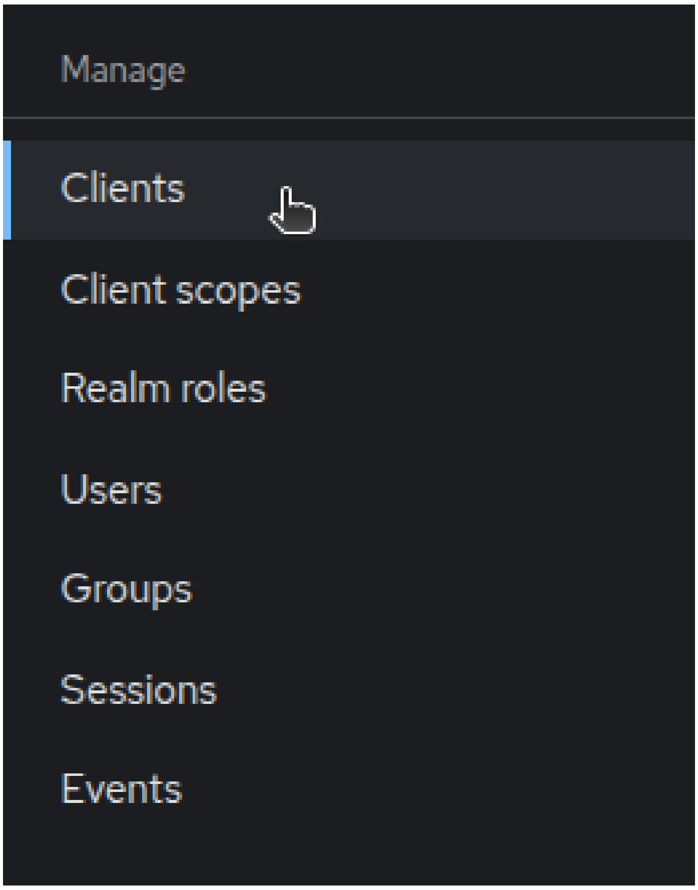


Fig. 28: Keycloak clients

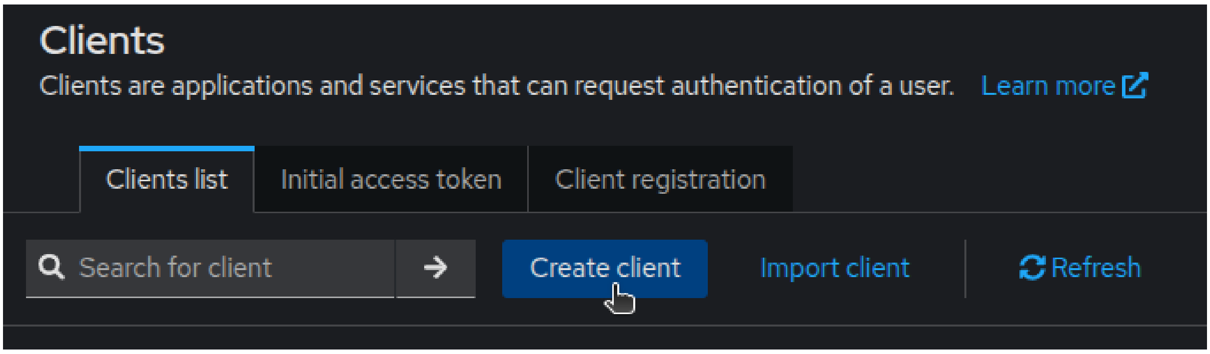


Fig. 29: Keycloak create client

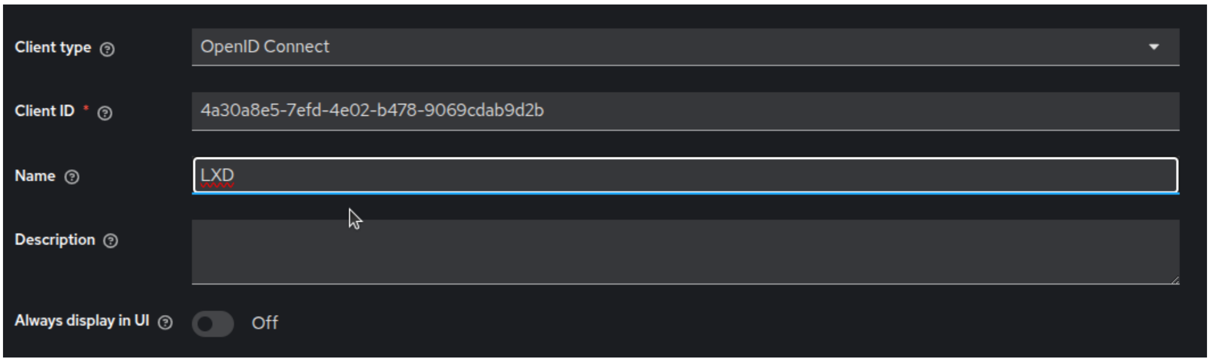


Fig. 30: Keycloak client name and ID

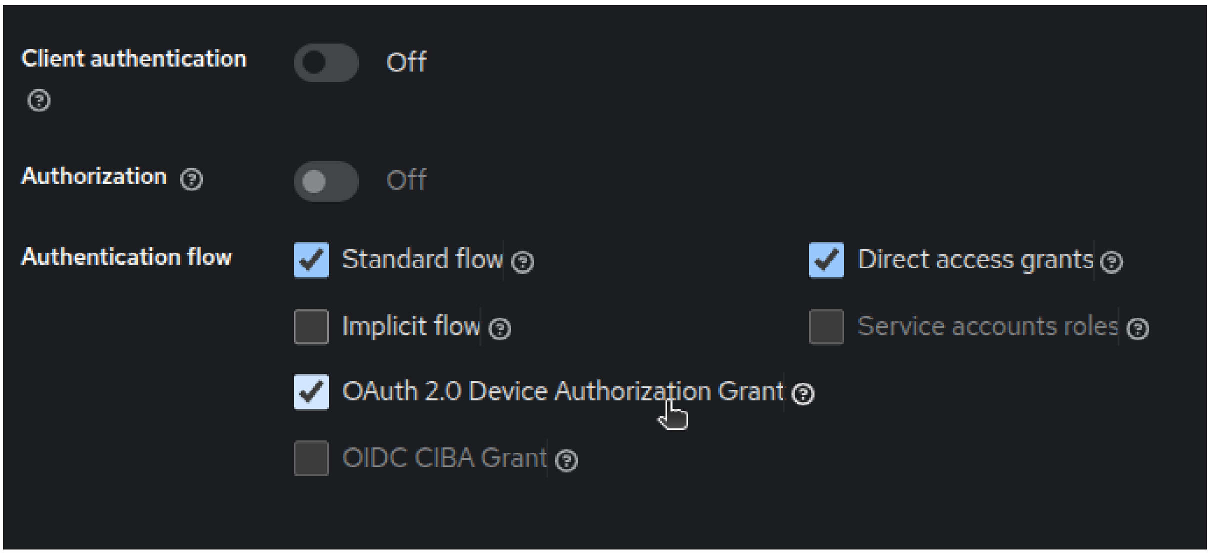


Fig. 31: Keycloak device flow

21. In Valid redirect URIs, type https://<your-LXD-hostname>/oidc/callback, then click Save.

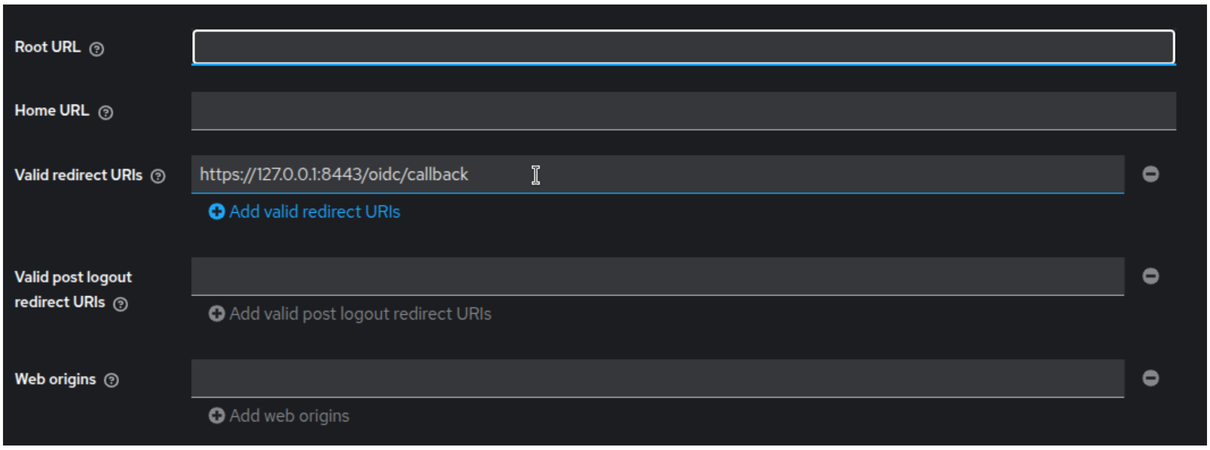


Fig. 32: Keycloak redirect URI

22. Go to Realm settings under Configure in the left panel.

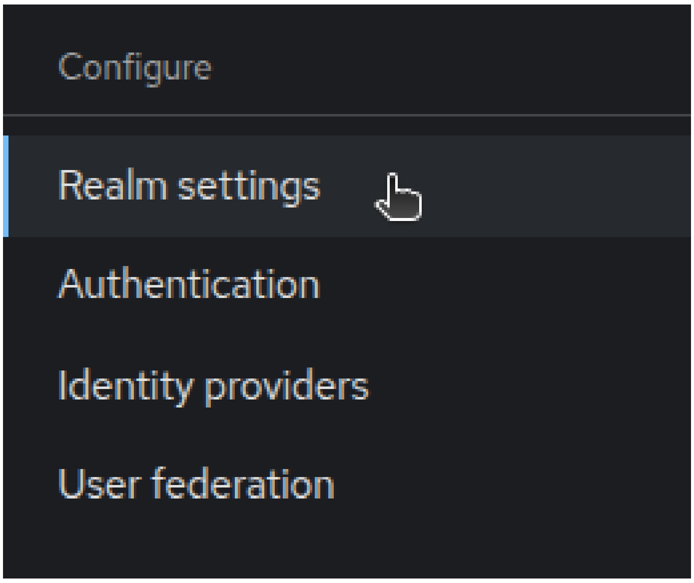


Fig. 33: Keycloak realm settings

23. Next to Endpoints, click OpenID Endpoint Configuration. This will display some output in JSON format. Copy the URL from the issuer field, and set this in LXD:

```
lxc config set oidc.issuer <your-issuer>
```

Alternatively, execute this command:

```
lxc config set oidc.issuer "$(curl <configuration-url> | jq -r .issuer)"
```

24. Configure LXD with the client ID that you configured in Keycloak in step 19.

```
lxc config set oidc.client.id <client-id>
```

You can now log in to LXD via the user interface or via the CLI. LXD will redirect you to Keycloak to authenticate. A Microsoft logo will be displayed that will, when clicked, allow you to log in to Keycloak (and therefore LXD) with Entra ID.

Additional Keycloak settings

It is important to remember that Keycloak is an identity provider in its own right. Once a user has signed in to Keycloak, information about that user is stored and a session is created. By default, even with a brokered identity provider, a user may edit their profile details on first log in. This includes editing their email address.

The information that Keycloak stores about the user is configurable in realm settings. When using Keycloak as a broker, you should consider preventing users from editing their information in Keycloak. It might be necessary to configure mappers[57] for the identity provider. Identity provider mappers configure Keycloak to automatically populate user profile information with fields from the brokered provider.

For more information on identity brokering with Keycloak, please see their documentation[58].

The following how-to guides cover common operations related to the LXD client (`lxc`):

How to add remote servers

> **Note**
>
> Remote servers are a concept in the LXD CLI.
>
> If you are using the UI or the API, you can interact with different remotes by using their exposed UI or API addresses instead.

By default, the command-line client interacts with the local LXD daemon, but you can add other servers or clusters to interact with.

One use case for remote servers is to distribute images that can be used to create instances on local servers. See *Remote image servers* (page 391) for more information.

You can also add a full LXD server as a remote server to your client. In this case, you can interact with the remote server in the same way as with your local daemon. For example, you can manage instances or update the server configuration on the remote server.

Authentication

To be able to add a LXD server as a remote server, the server's API must be exposed, which means that its *`core.https_address`* (page 402) server configuration option must be set.

When adding the server, you must then authenticate with it using the chosen method for *Remote API authentication* (page 358).

See *How to expose LXD to the network* (page 44) for more information.

[57] https://www.keycloak.org/docs/latest/server_admin/index.html#_mappers
[58] https://www.keycloak.org/docs/latest/server_admin/index.html#_identity_broker

List configured remotes

To see all configured remote servers, enter the following command:

```
lxc remote list
```

Remote servers that use the simple streams format[59] are pure image servers. Servers that use the `lxd` format are LXD servers, which either serve solely as image servers or might provide some images in addition to serving as regular LXD servers. See *Remote server types* (page 391) for more information.

Add a remote LXD server

To add a LXD server as a remote, enter the following command:

```
lxc remote add <remote_name> <IP|FQDN|URL|token> [flags]
```

Some authentication methods require specific flags (for example, use *`lxc remote add <remote_name> <IP|FQDN|URL> --auth-type=oidc`* (page 872) for OIDC authentication). See *Authenticate with the LXD server* (page 45) and *Remote API authentication* (page 358) for more information.

For example, enter the following command to add a remote through an IP address:

```
lxc remote add my-remote 192.0.2.10
```

You are prompted to confirm the remote server fingerprint and then asked for the token.

Select a default remote

The LXD command-line client is pre-configured with the `local` remote, which is the local LXD daemon.

To select a different remote as the default remote, enter the following command:

```
lxc remote switch <remote_name>
```

To see which server is configured as the default remote, enter the following command:

```
lxc remote get-default
```

Configure a global remote

You can configure remotes on a global, per-system basis. These remotes are available for every user of the LXD server for which you add the configuration.

Users can override these system remotes (for example, by running *`lxc remote rename`* (page 874) or *`lxc remote set-url`* (page 875)), which results in the remote and its associated certificates being copied to the user configuration.

To configure a global remote, edit the `config.yml` file that is located in one of the following directories:

- the directory specified by `LXD_GLOBAL_CONF` (if defined)

[59] https://git.launchpad.net/simplestreams/tree/

- /var/snap/lxd/common/global-conf/ (if you use the snap)
- /etc/lxd/ (otherwise)

Certificates for the remotes must be stored in the servercerts directory in the same location (for example, /etc/lxd/servercerts/). They must match the remote name (for example, foo.crt).

See the following example configuration:

```
remotes:
  foo:
    addr: https://192.0.2.4:8443
    auth_type: tls
    project: default
    protocol: lxd
    public: false
  bar:
    addr: https://192.0.2.5:8443
    auth_type: tls
    project: default
    protocol: lxd
    public: false
```

How to add command aliases

Note

Command aliases are a concept in the LXD CLI. They are not applicable to the UI or API.

The LXD command-line client supports adding aliases for commands that you use frequently. You can use aliases as shortcuts for longer commands, or to automatically add flags to existing commands.

To manage command aliases, you use the *lxc alias* (page 72) command.

For example, to always ask for confirmation when deleting an instance, create an alias for lxc delete that always runs lxc delete -i:

```
lxc alias add delete "delete -i"
```

To see all configured aliases, run *lxc alias list* (page 693). Run *lxc alias --help* (page 72) to see all available subcommands.

Related topics

Explanation:

- *About lxd and lxc* (page 345)
- *The LXD Dqlite database* (page 356)

Reference:

- *Architectures* (page 386)

- *Server configuration* (page 401)
- *REST API* (page 618)

2.2. Work with LXD

After the initial setup, you can start working with LXD by creating instances. You'll also need to set up and configure other entities.

2.2.1. Instances

The following how-to guides cover common operations related to instances.

How to create and manage instances:

How to create instances

When creating an instance, you must specify the *image* (page 348) on which the instance should be based.

Images contain a basic operating system (for example, a Linux distribution) and some LXD-related information. Images for various operating systems are available on the built-in remote image servers. See *Images* (page 148) for more information.

If you don't specify a name for the instance, LXD will automatically generate one. Instance names must be unique within a LXD deployment (also within a cluster). See *Instance name requirements* (page 415) for additional requirements.

CLI

API

UI

To create an instance, you can use either the `lxc init` (page 783) or the `lxc launch` (page 784) command. The `lxc init` (page 783) command only creates the instance, while the `lxc launch` (page 784) command creates and starts it.

Enter the following command to create a container:

```
lxc launch|init <image_server>:<image_name> <instance_name> [flags]
```

Unless the image is available locally, you must specify the name of the image server and the name of the image (for example, `ubuntu:24.04` for the official Ubuntu 24.04 LTS image).

See `lxc launch --help` (page 784) or `lxc init --help` (page 783) for a full list of flags. The most common flags are:

- `--config` to specify a configuration option for the new instance
- `--device` to override *device options* (page 447) for a device provided through a profile, or to specify an *initial configuration for the root disk device* (page 480) (syntax: `--device <device_name>,<device_option>=<value>`)
- `--profile` to specify a *profile* (page 97) to use for the new instance
- `--network` or `--storage` to make the new instance use a specific network or storage pool
- `--target` to create the instance on a specific cluster member
- `--vm` to create a virtual machine instead of a container

Instead of specifying the instance configuration as flags, you can pass it to the command as a YAML file.

For example, to launch a container with the configuration from `config.yaml`, enter the following command:

```
lxc launch ubuntu:24.04 ubuntu-config < config.yaml
```

Tip

Check the contents of an existing instance configuration (*lxc config show <instance_name> --expanded* (page 748)) to see the required syntax of the YAML file.

To create an instance, send a POST request to the `/1.0/instances` endpoint:

```
lxc query --request POST /1.0/instances --data '{
  "name": "<instance_name>",
  "source": {
    "alias": "<image_alias>",
    "protocol": "simplestreams",
    "server": "<server_URL>",
    "type": "image"
  }
}'
```

The return value of this query contains an operation ID, which you can use to query the status of the operation:

```
lxc query --request GET /1.0/operations/<operation_ID>
```

Use the following query to monitor the state of the instance:

```
lxc query --request GET /1.0/instances/<instance_name>/state
```

See `POST /1.0/instances` and `GET /1.0/instances/{name}/state` for more information.

The request creates the instance, but does not start it. To start an instance, send a PUT request to change the instance state:

```
lxc query --request PUT /1.0/instances/<instance_name>/state --data '{"action":
"start"}'
```

See *Start an instance* (page 93) for more information.

If you would like to start the instance upon creation, set the `start` property to true. The following example will create the container, then start it:

```
lxc query --request POST /1.0/instances --data '{
  "name": "<instance_name>",
  "source": {
    "alias": "<image_alias>",
    "protocol": "simplestreams",
```

(continues on next page)

```
    "server": "<server_URL>",
    "type": "image"
  },
  "start": true
}'
```

To create an instance, go to the *Instances* section and click *Create instance*.

On the resulting screen, optionally enter a name and description for the instance. Then click *Browse images* to select the image to be used for the instance. Depending on the selected image, you might be able to select the *instance type* (page 347) (container or virtual machine). You can also specify one or more profiles to use for the instance.

To further tweak the instance configuration or add devices to the instance, go to any of the tabs under *Advanced*. You can also edit the full instance configuration on the *YAML configuration* tab.

Finally, click *Create* or *Create and start* to create the instance.

Examples

The following CLI and API examples create the instances, but don't start them. If you are using the CLI client, you can use `lxc launch` (page 784) instead of `lxc init` (page 783) to automatically start them after creation.

In the UI, you can choose between *Create* and *Create and start* when you are ready to create the instance.

Create a container

To create a container with an Ubuntu 24.04 LTS image from the `ubuntu` server using the instance name `ubuntu-container`:

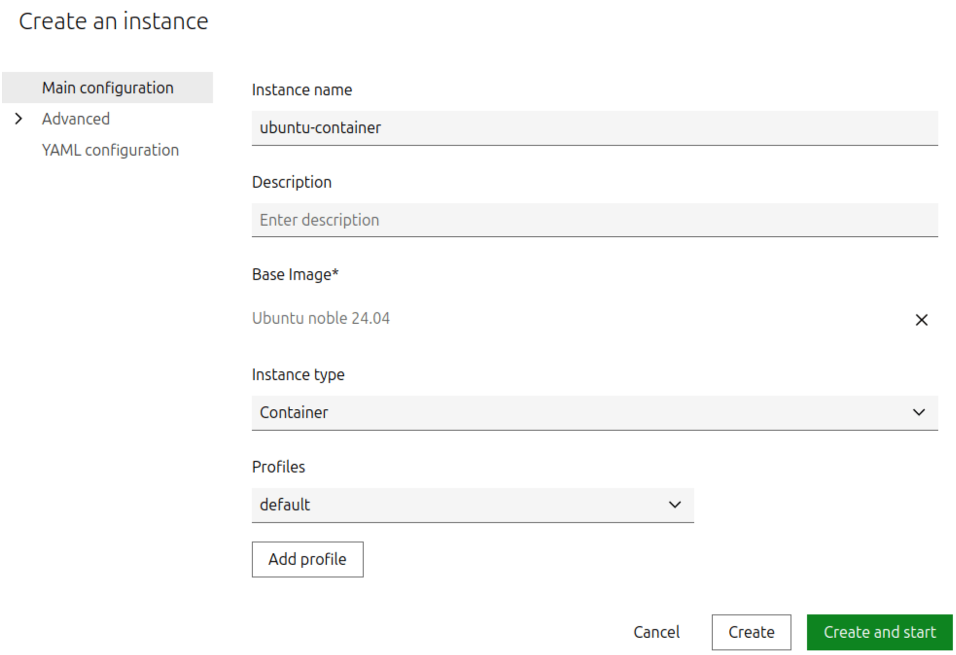
CLI

API

UI

```
lxc init ubuntu:24.04 ubuntu-container
```

```
lxc query --request POST /1.0/instances --data '{
  "name": "ubuntu-container",
  "source": {
    "alias": "24.04",
    "protocol": "simplestreams",
    "server": "https://cloud-images.ubuntu.com/releases/",
    "type": "image"
  }
}'
```



Create a virtual machine

To create a virtual machine with an Ubuntu 24.04 LTS image from the `ubuntu` server using the instance name `ubuntu-vm`:

CLI

API

UI

```
lxc init ubuntu:24.04 ubuntu-vm --vm
```

```
lxc query --request POST /1.0/instances --data '{
  "name": "ubuntu-vm",
  "source": {
    "alias": "24.04",
    "protocol": "simplestreams",
    "server": "https://cloud-images.ubuntu.com/releases/",
    "type": "image"
  },
  "type": "virtual-machine"
}'
```

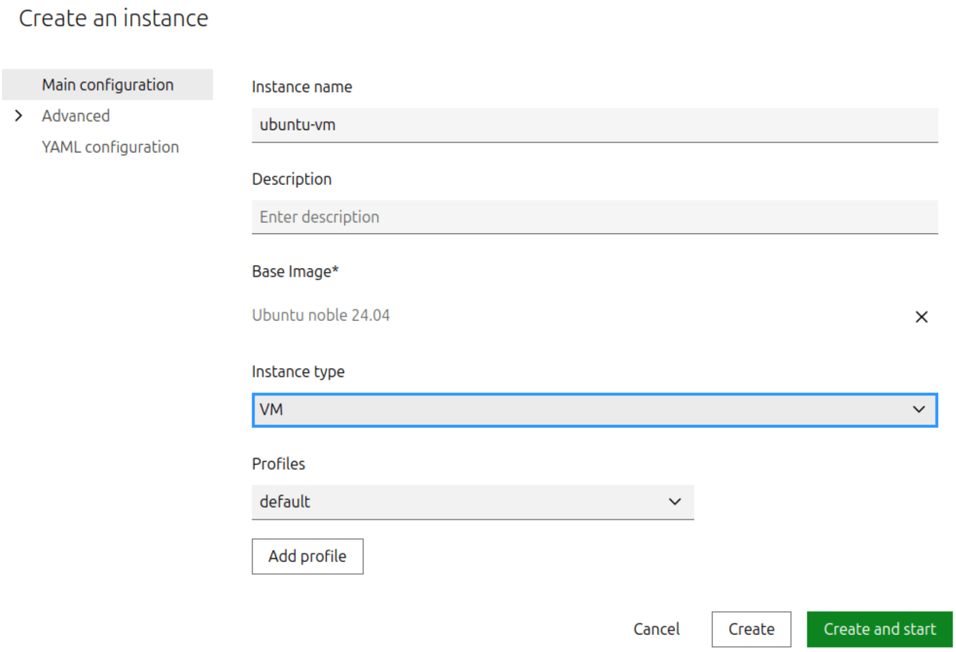
Or with a bigger disk:

CLI

API

UI

```
lxc init ubuntu:24.04 ubuntu-vm-big --vm --device root,size=30GiB
```




```
lxc query --request POST /1.0/instances --data '{
  "devices": {
    "root": {
      "path": "/",
      "pool": "default",
      "size": "30GiB",
      "type": "disk"
    }
  },
  "name": "ubuntu-vm-big",
  "source": {
    "alias": "24.04",
    "protocol": "simplestreams",
    "server": "https://cloud-images.ubuntu.com/releases/",
    "type": "image"
  },
  "type": "virtual-machine"
}'
```

Create a container with specific configuration options

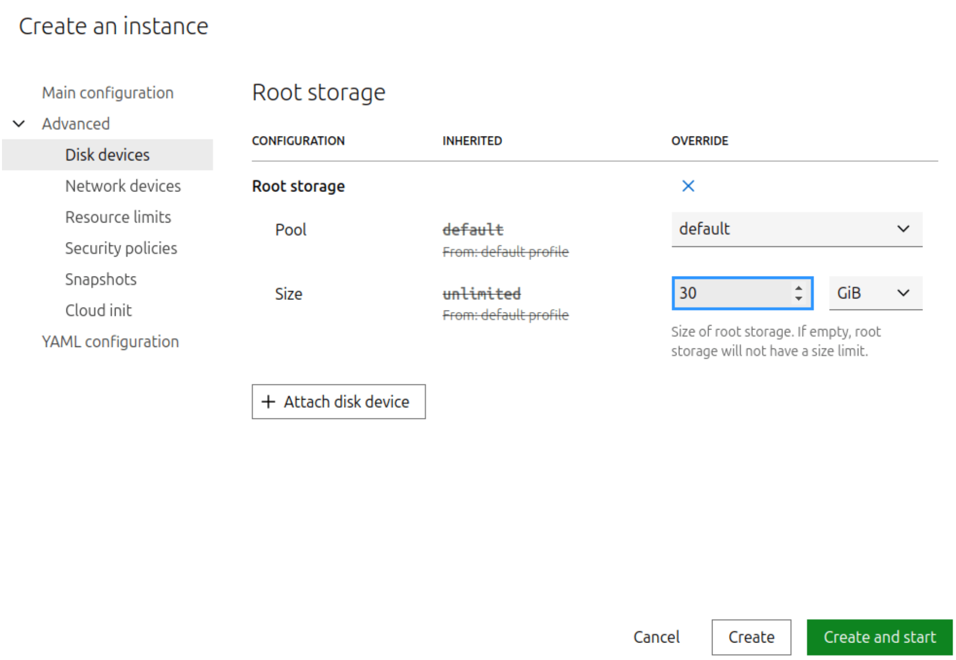
To create a container and limit its resources to one vCPU and 8 GiB of RAM:

CLI

API

UI

```
lxc init ubuntu:24.04 ubuntu-limited --config limits.cpu=1 --config limits.
memory=8GiB
```




```
lxc query --request POST /1.0/instances --data '{
  "config": {
    "limits.cpu": "1",
    "limits.memory": "8GiB"
  },
  "name": "ubuntu-limited",
  "source": {
    "alias": "24.04",
    "protocol": "simplestreams",
    "server": "https://cloud-images.ubuntu.com/releases/",
    "type": "image"
  }
}'
```

Create a VM on a specific cluster member

To create a virtual machine on the cluster member `micro2`, enter the following command:

CLI

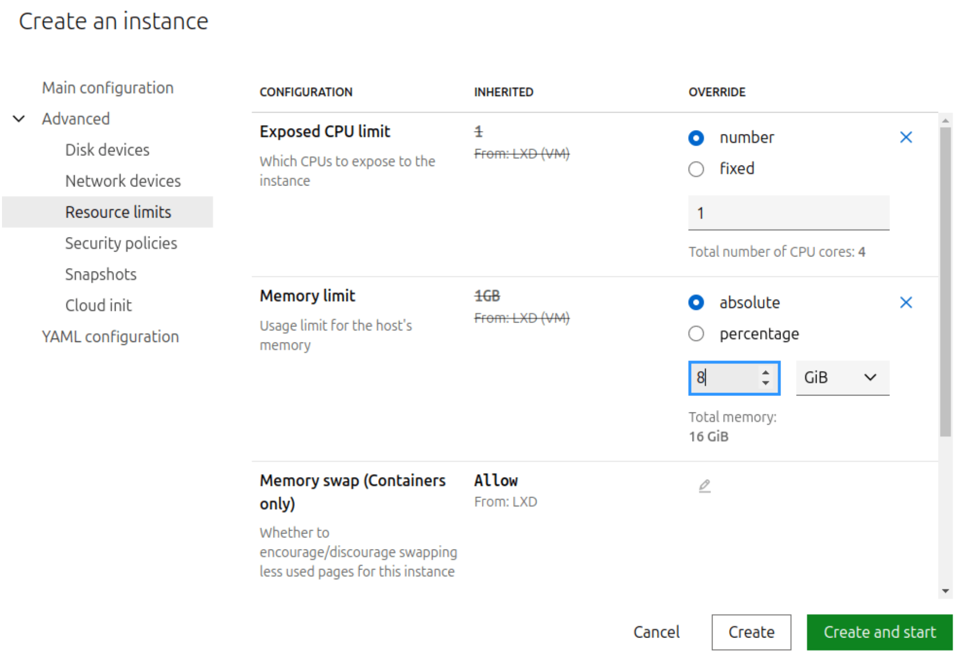
API

UI

```
lxc init ubuntu:24.04 ubuntu-vm-server2 --vm --target micro2
```

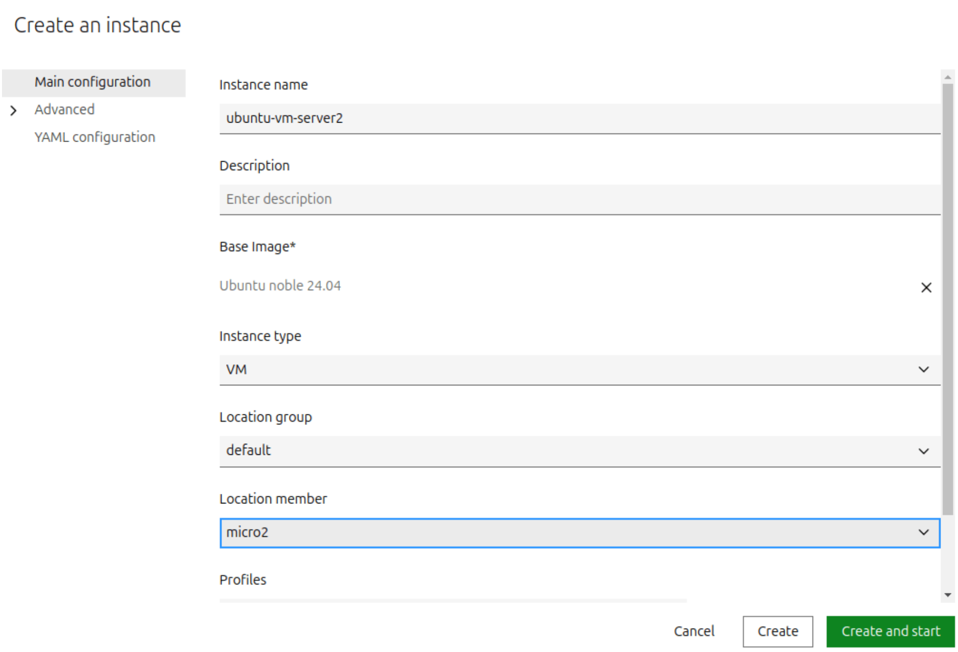
```
lxc query --request POST /1.0/instances?target=micro2 --data '{
  "name": "ubuntu-vm-server2",
  "source": {
    "alias": "24.04",
    "protocol": "simplestreams",
```

(continues on next page)



(continued from previous page)

```
    "server": "https://cloud-images.ubuntu.com/releases/",
    "type": "image"
  },
  "type": "virtual-machine"
}'
```



Create a container with a specific instance type

LXD supports simple instance types for clouds. Those are represented as a string that can be passed at instance creation time.

The list of supported clouds and instance types can be found at `images.lxd.canonical.com/meta/instance-types/all.yaml`[60].

The syntax allows the three following forms:

- `<instance type>`
- `<cloud>:<instance type>`
- `c<CPU>-m<RAM in GiB>`

For example, the following three instance types are equivalent:

- `t2.micro`
- `aws:t2.micro`
- `c1-m1`

To create a container with this instance type:

CLI

API

UI

```
lxc init ubuntu:24.04 my-instance --type t2.micro
```

```
lxc query --request POST /1.0/instances --data '{
  "instance_type": "t2.micro",
  "name": "my-instance",
  "source": {
    "alias": "24.04",
    "protocol": "simplestreams",
    "server": "https://cloud-images.ubuntu.com/releases/",
    "type": "image"
  }
}'
```

Creating an instance with a specific cloud instance type is currently not possible through the UI. Configure the corresponding options manually or through a profile.

Create a VM that boots from an ISO

To create a VM that boots from an ISO:

CLI

API

UI

First, create an empty VM that we can later install from the ISO image:

[60] https://images.lxd.canonical.com/meta/instance-types/all.yaml

```
lxc init iso-vm --empty --vm --config limits.cpu=2 --config limits.memory=4GiB --
device root,size=30GiB
```

Note

Adapt the `limits.cpu`, `limits.memory`, and root size based on the hardware recommendations for the ISO image used.

The second step is to import an ISO image that can later be attached to the VM as a storage volume:

```
lxc storage volume import <pool> <path-to-image.iso> iso-volume --type=iso
```

Lastly, attach the custom ISO volume to the VM using the following command:

```
lxc config device add iso-vm iso-volume disk pool=<pool> source=iso-volume boot.
priority=10
```

The *boot.priority* (page 480) configuration key ensures that the VM will boot from the ISO first. Start the VM and *connect to the console* (page 109) as there might be a menu you need to interact with:

```
lxc start iso-vm --console
```

Once you're done in the serial console, disconnect from the console using `Ctrl+a q` and *connect to the VGA console* (page 109) using the following command:

```
lxc console iso-vm --type=vga
```

You should now see the installer. After the installation is done, detach the custom ISO volume:

```
lxc storage volume detach <pool> iso-volume iso-vm
```

Now the VM can be rebooted, and it will boot from disk.

Note

On Linux virtual machines, the *LXD agent can be manually installed* (page 83).

First, create an empty VM that we can later install from the ISO image:

```
lxc query --request POST /1.0/instances --data '{
  "name": "iso-vm",
  "config": {
    "limits.cpu": "2",
    "limits.memory": "4GiB"
  },
  "devices": {
    "root": {
```

(continues on next page)

(continued from previous page)

```
      "path": "/",
      "pool": "default",
      "size": "30GiB",
      "type": "disk"
    }
  },
  "source": {
    "type": "none"
  },
  "type": "virtual-machine"
}'
```

Note

Adapt the values for `limits.cpu`, `limits.memory`, and `root: size` based on the hardware recommendations for the ISO image used.

The second step is to import an ISO image that can later be attached to the VM as a storage volume:

```
curl -X POST -H "Content-Type: application/octet-stream" -H "X-LXD-name: iso-
volume" \
-H "X-LXD-type: iso" --data-binary @<path-to-image.iso> --unix-socket /var/snap/
lxd/common/lxd/unix.socket \
lxd/1.0/storage-pools/<pool>/volumes/custom
```

Note

When importing an ISO image, you must send both binary data from a file and additional headers. The *lxc query* (page 869) command cannot do this, so you need to use `curl` or another tool instead.

Lastly, attach the custom ISO volume to the VM using the following command:

```
lxc query --request PATCH /1.0/instances/iso-vm --data '{
  "devices": {
    "iso-volume": {
      "boot.priority": "10",
      "pool": "<pool>",
      "source": "iso-volume",
      "type": "disk"
    }
  }
}'
```

The *boot.priority* (page 480) configuration key ensures that the VM will boot from the ISO first. Start the VM and *connect to the console* (page 109) as there might be a menu you need to interact with:

```
lxc query --request PUT /1.0/instances/iso-vm/state --data '{"action": "start"}'
lxc query --request POST /1.0/instances/iso-vm/console --data '{
  "height": 24,
  "type": "console",
  "width": 80
}'
```

Once you're done in the serial console, disconnect from the console using `Ctrl+a q` and *connect to the VGA console* (page 109) using the following command:

```
lxc query --request POST /1.0/instances/iso-vm/console --data '{
  "height": 24,
  "type": "vga",
  "width": 80
}'
```

You should now see the installer. After the installation is done, detach the custom ISO volume:

```
lxc query --request GET /1.0/instances/iso-vm
lxc query --request PUT /1.0/instances/iso-vm --data '{
  [...]
  "devices": {}
  [...]
}'
```

> **Note**
>
> You cannot remove the device through a PATCH request, but you must use a PUT request. Therefore, get the current configuration first and then provide the relevant configuration with an empty devices list through the PUT request.

Now the VM can be rebooted, and it will boot from disk.

```
   :end-before:
```

In the *Create instance* dialog, click *Use custom ISO* instead of *Browse images*. You can then upload your ISO file and install a VM from it.

Install the LXD agent into virtual machine instances

In order for features like direct command execution (`lxc exec` & `lxc shell`), file transfers (`lxc file`) and detailed usage metrics (`lxc info`) to work properly with virtual machines, an agent software is provided by LXD.

The virtual machine images from the official *remote image servers* (page 391) are pre-configured to load that agent on startup.

For other virtual machines, you may want to manually install the agent.

> **Note**
>
> The LXD agent is currently available only on Linux virtual machines using `systemd`.

LXD provides the agent through a remote `9p` file system and a `virtiofs` one that are both available under the mount name `config`. To install the agent, you'll need to get access to the virtual machine and run the following commands as root:

```
modprobe 9pnet_virtio
mount -t 9p config /mnt -o access=0,transport=virtio || mount -t virtiofs config /
mnt
cd /mnt
./install.sh
cd /
umount /mnt
reboot
```

You need to perform this task once.

Create a Windows VM

To create a Windows VM, you must first prepare a Windows image. See *Repack a Windows image* (page 161).

The How to install a Windows 11 VM using LXD[61] tutorial shows how to prepare the image and create a Windows VM from it.

How to configure instances

You can configure instances by setting *Instance properties* (page 415), *Instance options* (page 415), or by adding and configuring *Devices* (page 447).

See the following sections for instructions.

> **Note**
>
> To store and reuse different instance configurations, use *profiles* (page 97).

Configure instance options

You can specify instance options when you *create an instance* (page 73). Alternatively, you can update the instance options after the instance is created.

CLI

API

UI

Use the *`lxc config set`* (page 747) command to update instance options. Specify the instance name and the key and value of the instance option:

[61] https://ubuntu.com/tutorials/how-to-install-a-windows-11-vm-using-lxd

```
lxc config set <instance_name> <option_key>=<option_value> <option_key>=<option_
value> ...
```

Send a PATCH request to the instance to update instance options. Specify the instance name and the key and value of the instance option:

```
lxc query --request PATCH /1.0/instances/<instance_name> --data '{
  "config": {
    "<option_key>": "<option_value>",
    "<option_key>": "<option_value>"
  }
}'
```

See `PATCH /1.0/instances/{name}` for more information.

To update instance options, go to the *Configuration* tab of the instance detail page and click *Edit instance*.

Find the configuration option that you want to update and change its value. Click *Save changes* to save the updated configuration.

To configure instance options that are not displayed in the UI, follow the instructions in *Edit the full instance configuration* (page 91).

See *Instance options* (page 415) for a list of available options and information about which options are available for which instance type.

For example, change the memory limit for your container:

CLI

API

UI

To set the memory limit to 8 GiB, enter the following command:

```
lxc config set my-container limits.memory=8GiB
```

To set the memory limit to 8 GiB, send the following request:

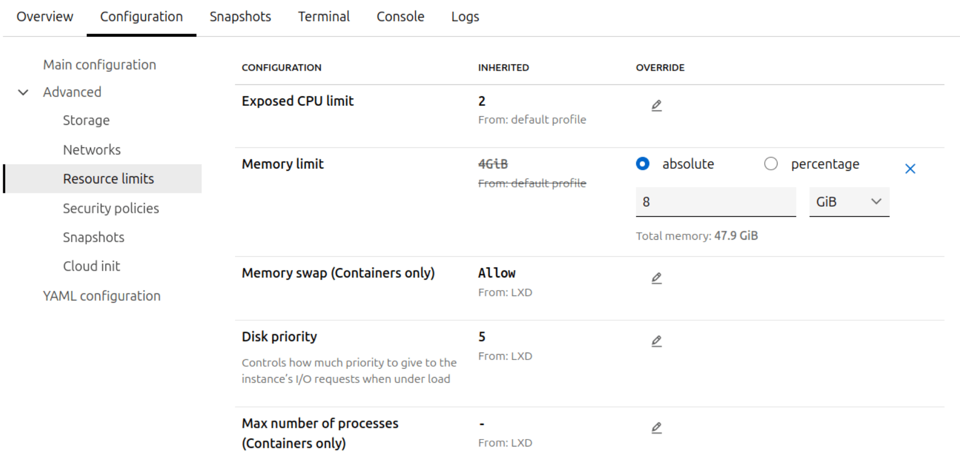
```
lxc query --request PATCH /1.0/instances/my-container --data '{
  "config": {
    "limits.memory": "8GiB"
  }
}'
```

To set the memory limit to 8 GiB, go to the *Configuration* tab of the instance detail page and select *Advanced > Resource limits*. Then click *Edit instance*.

Select *Override* for the **Memory limit** and enter 8 GiB as the absolute value.

Note

Some of the instance options are updated immediately while the instance is running. Others are updated only when the instance is restarted.



See the "Live update" information in the *Instance options* (page 415) reference for information about which options are applied immediately while the instance is running.

Configure instance properties

CLI

API

UI

To update instance properties after the instance is created, use the *lxc config set* (page 747) command with the `--property` flag. Specify the instance name and the key and value of the instance property:

```
lxc config set <instance_name> <property_key>=<property_value> <property_key>=
<property_value> ... --property
```

Using the same flag, you can also unset a property just like you would unset a configuration option:

```
lxc config unset <instance_name> <property_key> --property
```

You can also retrieve a specific property value with:

```
lxc config get <instance_name> <property_key> --property
```

To update instance properties through the API, use the same mechanism as for configuring instance options. The only difference is that properties are on the root level of the configuration, while options are under the `config` field.

Therefore, to set an instance property, send a PATCH request to the instance:

```
lxc query --request PATCH /1.0/instances/<instance_name> --data '{
  "<property_key>": "<property_value>",
  "<property_key>": "property_value>"
  }
}'
```

To unset an instance property, send a PUT request that contains the full instance configuration that you want except for the property that you want to unset.

See `PATCH /1.0/instances/{name}` and `PUT /1.0/instances/{name}` for more information.

The LXD UI does not distinguish between instance options and instance properties. Therefore, you can configure instance properties in the same way as you *configure instance options* (page 84).

Configure devices

Generally, devices can be added or removed for a container while it is running. VMs support hotplugging for some device types, but not all.

See *Devices* (page 447) for a list of available device types and their options.

> **Note**
>
> Every device entry is identified by a name unique to the instance.
>
> Devices from profiles are applied to the instance in the order in which the profiles are assigned to the instance. Devices defined directly in the instance configuration are applied last. At each stage, if a device with the same name already exists from an earlier stage, the whole device entry is overridden by the latest definition.
>
> Device names are limited to a maximum of 64 characters.

CLI

API

UI

To add and configure an instance device for your instance, use the *`lxc config device add`* (page 739) command.

Specify the instance name, a device name, the device type and maybe device options (depending on the *device type* (page 447)):

```
lxc config device add <instance_name> <device_name> <device_type> <device_option_key>=<device_option_value> <device_option_key>=<device_option_value> ...
```

For example, to add the storage at `/share/c1` on the host system to your instance at path `/opt`, enter the following command:

```
lxc config device add my-container disk-storage-device disk source=/share/c1 path=/opt
```

To configure instance device options for a device that you have added earlier, use the *`lxc config device set`* (page 742) command:

```
lxc config device set <instance_name> <device_name> <device_option_key>=<device_option_value> <device_option_key>=<device_option_value> ...
```

Device options for a device inherited from a profile cannot be updated within the instance. Use the *`lxc config device override`* (page 741) command to create a copy of the profile device with updated device options. The newly created instance device will override the inherited device.

Specify the instance name, device name and the device options that should be overridden:

```
lxc config device override <instance_name> <device_name> <device_option_key>=
<device_option_value> <device_option_key>=<device_option_value> ...
```

Note

You can also specify device options by using the `--device` flag when *creating an instance* (page 73). This is useful if you want to override device options for a device that is provided through a *profile* (page 97).

To remove a device, use the *`lxc config device remove`* (page 742) command. See *`lxc config device --help`* (page 738) for a full list of available commands.

To add or configure an instance device for your instance, use the same mechanism of patching the instance configuration. The device configuration is located under the `devices` field of the configuration.

Caution

Patching a device's configuration unsets any omitted options for that device, along with the instance's `description` property. See *Effects of patching device options* (page 89) for details.

Specify the instance name, a device name, and any *Required device options* (page 89) (depending on the *device type* (page 447)):

```
lxc query --request PATCH /1.0/instances/<instance_name> --data '{
  "devices": {
    "<device_name>": {
      "type": "<device_type>",
      "<device_option_key>": "<device_option_value>",
      "<device_option_key>": "device_option_value>"
    }
  }
}'
```

For example, to add the storage at `/share/c1` on the host system to your instance at path `/opt`, enter the following command:

```
lxc query --request PATCH /1.0/instances/my-container --data '{
  "devices": {
    "disk-storage-device": {
      "type": "disk",
      "source": "/share/c1",
      "path": "/opt"
    }
```

(continues on next page)

```
  }
}'
```

See `PATCH /1.0/instances/{name}` for more information.

Required device options

When using a PATCH request to update an instance's `devices` property, you must include any required options for each device in the request body. The device's `type` option is always required. To find any other required keys for a specific device type, view the *Devices* (page 447) reference guides. For example, for an OVN NIC device, the *`network`* (page 465) key is required.

Effects of patching device options

For any device in your PATCH request, the request acts similar to a conventional PUT: it replaces all options for that device. This means that if you omit a non-required option, it is unset. Thus, include not only the options you want to add or update in your patch, but also any other existing options whose values you want to keep.

This behavior only affects the specific device or devices that you are patching; if there are other devices, you don't need to include them. It also does not affect any other instance properties, with one exception: if the instance includes a `description` property, that property must be passed along with `devices`; otherwise, it is unset.

For example, consider an instance that contains this `devices` property:

```
"devices": {
  "my-bridge-nic": {
    "name": "my-bridge-nic-name",
    "network": "my-bridge-network",
    "type": "nic"
  },
  "my-ovn-nic": {
    "name": "my-ovn-nic-name",
    "network": "my-ovn-network",
    "type": "nic"
  }
}
```

Let's say the following PATCH request is sent for this instance:

```
lxc query --request PATCH /1.0/instances/my-instance --data '{
  "devices": {
    "my-bridge-nic": {
      "type": "nic",
      "network": "test-bridge",
      "ipv4.address": "192.0.2.10"
    }
  }
}'
```

This PATCH request updates only the `my-bridge-nic` device, without affecting the `my-ovn-nic` device. The device options defined in the request body replace the existing options. After the request, this is the `devices` property's configuration:

```
"devices": {
  "my-bridge-nic": {
    "network": "my-bridge-network",
    "type": "nic",
    "ipv4.address": "192.0.2.10"
  },
  "my-ovn-nic": {
    "name": "my-ovn-nic-name",
    "network": "my-ovn-network",
    "type": "nic"
  }
}
```

Notice that in the updated `my-bridge-nic` device, the `name` option is unset and no longer appears, due to not being sent in the PATCH request.

The UI does not support all device types yet, but you can configure disk and network devices for your instances.

To attach a device to your instance, or modify an existing device, update your instance configuration (in the same way as you *configure instance options* (page 84)). Select *Advanced* > *Disk devices* > *Attach disk device* or *Advanced* > *Network devices* > *Attach network* to create a device and attach it to your instance.

Note

Some of the devices that are displayed in the instance configuration are inherited from a *profile* (page 97) or defined through a *project* (page 161). Depending on the type of device, it might not be possible to edit these devices for an instance.

To add and configure devices that are not currently supported in the UI, follow the instructions in *Edit the full instance configuration* (page 91).

Display instance configuration

CLI

API

UI

To display the current configuration of your instance, including writable instance properties, instance options, devices and device options, enter the following command:

```
lxc config show <instance_name> --expanded
```

To retrieve the current configuration of your instance, including writable instance properties, instance options, devices and device options, send a GET request to the instance:

```
lxc query --request GET /1.0/instances/<instance_name>
```

See GET /1.0/instances/{name} for more information.

To view the current configuration of your instance, go to *Instances*, select your instance, and then switch to the *Configuration* tab.

To see the full configuration including instance properties, instance options, devices and device options (also the ones that aren't yet supported by the UI), select *YAML configuration*. This view shows the full YAML of the instance configuration.

Edit the full instance configuration

CLI

API

UI

To edit the full instance configuration, including writable instance properties, instance options, devices and device options, enter the following command:

```
lxc config edit <instance_name>
```

> **Note**
>
> For convenience, the *lxc config edit* (page 744) command displays the full configuration including read-only instance properties. However, you cannot edit those properties. Any changes are ignored.

To update the full instance configuration, including writable instance properties, instance options, devices and device options, send a PUT request to the instance:

```
lxc query --request PUT /1.0/instances/<instance_name> --data '<instance_
configuration>'
```

See PUT /1.0/instances/{name} for more information.

> **Note**
>
> If you include changes to any read-only instance properties in the configuration you provide, they are ignored.

Instead of using the UI forms to configure your instance, you can choose to edit the YAML configuration of the instance. You must use this method if you need to update any configurations that are not available in the UI.

> **Important**
>
> When doing updates, do not navigate away from the YAML configuration without saving your changes. If you do, your updates are lost.

To edit the YAML configuration of your instance, go to the instance detail page, switch to the *Configuration* tab and select *YAML configuration*. Then click *Edit instance*.

Edit the YAML configuration as required. Then click *Save changes* to save the updated configuration.

Note

For convenience, the YAML contains the full configuration including read-only instance properties. However, you cannot edit those properties. Any changes are ignored.

How to manage instances

When listing the existing instances, you can see their type, status, and location (if applicable). You can filter the instances and display only the ones that you are interested in.

CLI

API

UI

Enter the following command to list all instances:

```
lxc list
```

You can filter the instances that are displayed, for example, by type, status or the cluster member where the instance is located:

```
lxc list type=container
lxc list status=running
lxc list location=server1
```

You can also filter by name. To list several instances, use a regular expression for the name. For example:

```
lxc list ubuntu.*
```

Enter *`lxc list --help`* (page 785) to see all filter options.

Query the `/1.0/instances` endpoint to list all instances. You can use *Recursion* (page 621) to display more information about the instances:

```
lxc query --request GET /1.0/instances?recursion=2
```

You can *filter* (page 621) the instances that are displayed, by name, type, status or the cluster member where the instance is located:

```
lxc query --request GET /1.0/instances?filter=name+eq+ubuntu
lxc query --request GET /1.0/instances?filter=type+eq+container
lxc query --request GET /1.0/instances?filter=status+eq+running
lxc query --request GET /1.0/instances?filter=location+eq+server1
```

To list several instances, use a regular expression for the name. For example:

```
lxc query --request GET /1.0/instances?filter=name+eq+ubuntu.*
```

See `GET /1.0/instances` for more information.

Go to *Instances* to see a list of all instances.

You can filter the instances that are displayed by status, instance type, or the profile they use by selecting the corresponding filter.

In addition, you can search for instances by entering a search text. The text you enter is matched against the name, the description, and the name of the base image.

Show information about an instance

CLI

API

UI

Enter the following command to show detailed information about an instance:

```
lxc info <instance_name>
```

Add `--show-log` to the command to show the latest log lines for the instance:

```
lxc info <instance_name> --show-log
```

Query the following endpoint to show detailed information about an instance:

```
lxc query --request GET /1.0/instances/<instance_name>
```

See `GET /1.0/instances/{name}` for more information.

Clicking an instance line in the overview will show a summary of the instance information right next to the instance list.

Click the instance name to go to the instance detail page, which contains detailed information about the instance.

Start an instance

CLI

API

UI

Enter the following command to start an instance:

```
lxc start <instance_name>
```

You will get an error if the instance does not exist or if it is running already.

To immediately attach to the console when starting, pass the `--console` flag. For example:

```
lxc start <instance_name> --console
```

See *How to access the console* (page 109) for more information.

To start an instance, send a PUT request to change the instance state:

```
lxc query --request PUT /1.0/instances/<instance_name>/state --data '{"action":
"start"}'
```

The return value of this query contains an operation ID, which you can use to query the status of the operation:

```
lxc query --request GET /1.0/operations/<operation_ID>
```

Use the following query to monitor the state of the instance:

```
lxc query --request GET /1.0/instances/<instance_name>/state
```

See `GET /1.0/instances/{name}/state` and `PUT /1.0/instances/{name}/state`for more information.

To start an instance, go to the instance list or the respective instance and click the *Start* button (□).

You can also start several instances at the same time by selecting them in the instance list and clicking the *Start* button at the top.

On the instance detail page, select the *Console* tab to see the boot log with information about the instance starting up. Once it is running, you can select the *Terminal* tab to access the instance.

Prevent accidental start of instances

To protect a specific instance from being started, set *security.protection.start* (page 437) to `true` for the instance. See *How to configure instances* (page 84) for instructions.

Stop an instance

CLI

API

UI

Enter the following command to stop an instance:

```
lxc stop <instance_name>
```

You will get an error if the instance does not exist or if it is not running.

To stop an instance, send a PUT request to change the instance state:

```
lxc query --request PUT /1.0/instances/<instance_name>/state --data '{"action":
"stop"}'
```

The return value of this query contains an operation ID, which you can use to query the status of the operation:

```
lxc query --request GET /1.0/operations/<operation_ID>
```

Use the following query to monitor the state of the instance:

```
lxc query --request GET /1.0/instances/<instance_name>/state
```

See `GET /1.0/instances/{name}/state` and `PUT /1.0/instances/{name}/state` for more information.

To stop an instance, go to the instance list or the respective instance and click the *Stop* button (). You are then prompted to confirm.

Tip

To skip the confirmation prompt, hold the `Shift` key while clicking.

You can choose to force-stop the instance. If stopping the instance takes a long time or the instance is not responding to the stop request, click the spinning stop button to go back to the confirmation prompt, where you can select to force-stop the instance.

You can also stop several instances at the same time by selecting them in the instance list and clicking the *Stop* button at the top.

Delete an instance

If you don't need an instance anymore, you can remove it. The instance must be stopped before you can delete it.

CLI

API

UI

Enter the following command to delete an instance:

```
lxc delete <instance_name>
```

To delete an instance, send a DELETE request to the instance:

```
lxc query --request DELETE /1.0/instances/<instance_name>
```

See `DELETE /1.0/instances/{name}` for more information.

To delete an instance, go to its instance detail page and click *Delete instance*. You are then prompted to confirm.

Tip

To skip the confirmation prompt, hold the `Shift` key while clicking.

You can also delete several instances at the same time by selecting them in the instance list and clicking the *Delete* button at the top.

Caution

This command permanently deletes the instance and all its snapshots.

Prevent accidental deletion of instances

There are different ways to prevent accidental deletion of instances:

- To protect a specific instance from being deleted, set *security.protection.delete* (page 436) to `true` for the instance. See *How to configure instances* (page 84) for instructions.
- In the CLI client, you can create an alias to be prompted for approval every time you use the *lxc delete* (page 762) command:

  ```
  lxc alias add delete "delete -i"
  ```

Rebuild an instance

If you want to wipe and re-initialize the root disk of your instance but keep the instance configuration, you can rebuild the instance.

Rebuilding is only possible for instances that do not have any snapshots.

Stop your instance before rebuilding it.

CLI

API

UI

Enter the following command to rebuild the instance with a different image:

```
lxc rebuild <image_name> <instance_name>
```

Enter the following command to rebuild the instance with an empty root disk:

```
lxc rebuild <instance_name> --empty
```

For more information about the `rebuild` command, see *lxc rebuild --help* (page 870).

To rebuild the instance with a different image, send a POST request to the instance's `rebuild` endpoint. For example:

```
lxc query --request POST /1.0/instances/<instance_name>/rebuild --data '{
  "source": {
    "alias": "<image_alias>",
    "protocol": "simplestreams",
    "server": "<server_URL>"
  }
}'
```

To rebuild the instance with an empty root disk, specify the source type as `none`:

```
lxc query --request POST /1.0/instances/<instance_name>/rebuild --data '{
  "source": {
    "type": "none"
  }
}'
```

See `POST /1.0/instances/{name}/rebuild` for more information.

Rebuilding an instance is not yet supported in the UI.

How to use profiles

Profiles store a set of configuration options. They can contain *Instance options* (page 415), *Devices* (page 447), and device options.

You can apply any number of profiles to an instance. They are applied in the order they are specified, so the last profile to specify a specific key takes precedence. However, instance-specific configuration always overrides the configuration coming from the profiles.

> **Note**
>
> Profiles can be applied to containers and virtual machines. Therefore, they might contain options and devices that are valid for either type.
>
> When applying a profile that contains configuration that is not suitable for the instance type, this configuration is ignored and does not result in an error.

If you don't specify any profiles when launching a new instance, the `default` profile is applied automatically. This profile defines a network interface and a root disk. The `default` profile cannot be renamed or removed.

View profiles

CLI

API

UI

Enter the following command to display a list of all available profiles:

```
lxc profile list
```

Enter the following command to display the contents of a profile:

```
lxc profile show <profile_name>
```

To display all available profiles, send a request to the `/1.0/profiles` endpoint:

```
lxc query --request GET /1.0/profiles?recursion=1
```

To display a specific profile, send a request to that profile:

```
lxc query --request GET /1.0/profiles/<profile_name>
```

See `GET /1.0/profiles` and `GET /1.0/profiles/{name}` for more information.

Go to the *Profiles* section to view all available profiles.

To view information about a specific profile, click its line in the overview. To display the full information about a profile, including its configuration, click the profile name to go to the profile detail page.

Create an empty profile

CLI

API

UI

Enter the following command to create an empty profile:

```
lxc profile create <profile_name>
```

To create an empty profile, send a POST request to the `/1.0/profiles` endpoint:

```
lxc query --request POST /1.0/profiles --data '{"name": "<profile_name>"}'
```

See `POST /1.0/profiles` for more information.

To create a profile, go to the *Profiles* section and click *Create profile*.

Enter at least a profile name and click *Create* to save the new profile.

Edit a profile

You can either set specific configuration options for a profile or edit the full profile. See *Instance configuration* (page 414) (and its subpages) for the available options.

Set specific options for a profile

CLI

API

UI

To set an instance option for a profile, use the *`lxc profile set`* (page 859) command. Specify the profile name and the key and value of the instance option:

```
lxc profile set <profile_name> <option_key>=<option_value> <option_key>=<option_
value> ...
```

To add and configure an instance device for your profile, use the *`lxc profile device add`* (page 852) command. Specify the profile name, a device name, the device type and maybe device options (depending on the *device type* (page 447)):

```
lxc profile device add <profile_name> <device_name> <device_type> <device_option_
key>=<device_option_value> <device_option_key>=<device_option_value> ...
```

To configure instance device options for a device that you have added to the profile earlier, use the *lxc profile device set* (page 855) command:

```
lxc profile device set <profile_name> <device_name> <device_option_key>=<device_
option_value> <device_option_key>=<device_option_value> ...
```

To set an instance option for a profile, send a PATCH request to the profile. Specify the key and value of the instance option under the "config" field:

```
lxc query --request PATCH /1.0/profiles/<profile_name> --data '{
  "config": {
    "<option_key>": "<option_value>",
    "<option_key>": "<option_value>"
  }
}'
```

To add and configure an instance device for your profile, specify the device name, the device type and maybe device options (depending on the *device type* (page 447)) under the "devices" field:

```
lxc query --request PATCH /1.0/profiles/<profile_name> --data '{
  "devices": {
    "<device_name>": {
      "type": "<device_type>",
      "<device_option_key>": "<device_option_value>",
      "<device_option_key>": "<device_option_value>"
    }
  }
}'
```

See `PATCH /1.0/profiles/{name}` for more information.

To configure a profile, select it from the *Profiles* overview, switch to the *Configuration* tab and click *Edit profile*. You can then configure options for the profile in the same way as you *configure instance options* (page 84).

Edit the full profile

Instead of setting each configuration option separately, you can provide all options at once.

Check the contents of an existing profile or instance configuration for the required fields. For example, the default profile might look like this:

```
config: {}
description: Default LXD profile
devices:
  eth0:
    name: eth0
    network: lxdbr0
    type: nic
  root:
    path: /
    pool: default
```

(continues on next page)

```
    type: disk
name: default
used_by:
```

Instance options are provided as an array under `config`. Instance devices and instance device options are provided under `devices`.

CLI

API

UI

To edit a profile using your standard terminal editor, enter the following command:

```
lxc profile edit <profile_name>
```

Alternatively, you can create a YAML file (for example, `profile.yaml`) with the configuration and write the configuration to the profile with the following command:

```
lxc profile edit <profile_name> < profile.yaml
```

To update the entire profile configuration, send a PUT request to the profile:

```
lxc query --request PUT /1.0/profiles/<profile_name> --data '{
  "config": { ... },
  "description": "<description>",
  "devices": { ... }
}'
```

See `PUT /1.0/profiles/{name}` for more information.

To edit the YAML configuration of a profile, go to the profile detail page, switch to the *Configuration* tab and select *YAML configuration*. Then click *Edit profile*.

Edit the YAML configuration as required. Then click *Save changes* to save the updated configuration.

Important

When doing updates, do not navigate away from the YAML configuration without saving your changes. If you do, your updates are lost.

Apply a profile to an instance

CLI

API

UI

Enter the following command to apply a profile to an instance:

```
lxc profile add <instance_name> <profile_name>
```

💡 **Tip**

Check the configuration after adding the profile: *lxc config show <instance_name>* (page 748)

You will see that your profile is now listed under `profiles`. However, the configuration options from the profile are not shown under `config` (unless you add the `--expanded` flag). The reason for this behavior is that these options are taken from the profile and not the configuration of the instance.

This means that if you edit a profile, the changes are automatically applied to all instances that use the profile.

You can also specify profiles when launching an instance by adding the `--profile` flag:

```
lxc launch <image> <instance_name> --profile <profile> --profile <profile> ...
```

To apply a profile to an instance, add it to the profile list in the instance configuration:

```
lxc query --request PATCH /1.0/instances/<instance_name> --data '{
  "profiles": [ "default", "<profile_name>" ]
}'
```

See `PATCH /1.0/instances/{name}` for more information.

You can also specify profiles when *creating an instance* (page 73):

```
lxc query --request POST /1.0/instances --data '{
  "name": "<instance_name>",
  "profiles": [ "default", "<profile_name>" ],
  "source": {
    "alias": "<image_alias>",
    "protocol": "simplestreams",
    "server": "<server_URL>",
    "type": "image"
  }
}'
```

To apply a profile to an instance, select the instance from the *Instances* overview, switch to the *Configuration* tab and click *Edit instance*. You can then select a profile from the drop-down list, or click *Add profile* to attach another profile in addition to the one (or more) that are already attached to the instance.

If you attach more than one profile to an instance, you can specify the order in which the profiles are applied by moving each profile up or down the list.

You can also apply profiles in the same way when *creating an instance* (page 73).

Remove a profile from an instance

CLI

API

UI

Enter the following command to remove a profile from an instance:

```
lxc profile remove <instance_name> <profile_name>
```

To remove a profile from an instance, send a PATCH request to the instance configuration with the new profile list. For example, to revert back to using only the default profile:

```
lxc query --request PATCH /1.0/instances/<instance_name> --data '{
  "profiles": [ "default" ]
}'
```

See `PATCH /1.0/instances/{name}` for more information.

To remove a profile from an instance, select the instance from the *Instances* overview, switch to the *Configuration* tab and click *Edit instance*. Click the *Delete* link next to a profile to remove it from the instance.

How to troubleshoot failing instances

If your instance fails to start and ends up in an error state, this usually indicates a bigger issue related to either the image that you used to create the instance or the server configuration.

To troubleshoot the problem, complete the following steps:

1. Save the relevant log files and debug information:

   Instance log

   Display the instance log:

   CLI

   API

   UI

   ```
   lxc info <instance_name> --show-log
   ```

   ```
   lxc query --request GET /1.0/instances/<instance_name>/logs/lxc.log
   ```

   Navigate to the instance detail page and switch to the *Logs* tab to view the available log files.

   Console log

   Display the console log:

   CLI

   API

   UI

   ```
   lxc console <instance_name> --show-log
   ```

   This command is available only for containers.

   ```
   lxc query --request GET /1.0/instances/<instance_name>/console
   ```

This endpoint is available only for containers.

Navigate to the instance detail page and switch to the *Console* tab to view the console. The console is displayed only when the instance is running.

Detailed server information

The LXD snap includes a tool that collects the relevant server information for debugging. Enter the following command to run it:

```
sudo lxd.buginfo
```

2. Reboot the machine that runs your LXD server.

3. Try starting your instance again. If the error occurs again, compare the logs to check if it is the same error.

   If it is, and if you cannot figure out the source of the error from the log information, open a question in the forum[62]. Make sure to include the log files you collected.

Troubleshooting examples

See the following sections for some typical methods of troubleshooting an instance.

Debug `systemd init`

Here is how to enable `systemd` debug level messages[63] for the `c1` container:

```
lxc config set c1 raw.lxc 'lxc.init.cmd = /sbin/init systemd.log_level=debug'

lxc start c1
```

Now that the container has started, you can check for the debug messages in the journal:

```
lxc exec c1 -- journalctl
```

Emergency `systemd` shell

Here is how to get an `emergency` shell on an instance using `systemd`:

```
lxc config set c1 raw.lxc 'lxc.init.cmd = /sbin/init emergency'

lxc start c1
```

Now that the container has started, you can enter the emergency shell using the console (hit the `Enter` key once in):

```
lxc console c1
```

[62] https://discourse.ubuntu.com/c/lxd/126
[63] https://www.freedesktop.org/wiki/Software/systemd/Debugging/

Issue starting RHEL 7 container

In this example, let's investigate a RHEL 7 system in which `systemd` cannot start.

```
~$ lxc console --show-log rhel7 Console log: Failed to insert module
'autofs4'Failed to insert module 'unix'Failed to mount sysfs at /sys:
Operation not permittedFailed to mount proc at /proc: Operation not
permitted[!!!!!!] Failed to mount API filesystems, freezing.
```

The errors here say that `/sys` and `/proc` cannot be mounted - which is correct in an unprivileged container. However, LXD mounts these file systems automatically if it can.

The *container requirements* (page 398) specify that every container must come with an empty `/dev`, `/proc` and `/sys` directory, and that `/sbin/init` must exist. If those directories don't exist, LXD cannot mount them, and `systemd` will then try to do so. As this is an unprivileged container, `systemd` does not have the ability to do this, and it then freezes.

So you can see the environment before anything is changed, and you can explicitly change the init system in a container using the *`raw.lxc`* (page 431) configuration parameter. This is equivalent to setting `init=/bin/bash` on the Linux kernel command line.

```
lxc config set rhel7 raw.lxc 'lxc.init.cmd = /bin/bash'
```

Here is what it looks like:

```
~$ lxc config set rhel7 raw.lxc 'lxc.init.cmd = /bin/bash' ~$ lxc start rhel7
~$ lxc console --show-log rhel7  Console log: [root@rhel7 /]#
```

Now that the container has started, you can check it and see that things are not running as well as expected:

```
~$ lxc exec rhel7 -- bash [root@rhel7 ~]# ls[root@rhel7 ~]# mountmount:
failed to read mtab: No such file or directory[root@rhel7 ~]# cd /[root@rhel7
/]# ls /proc/sys[root@rhel7 /]# exit
```

Because LXD tries to auto-heal, it created some of the directories when it was starting up. Shutting down and restarting the container fixes the problem, but the original cause is still there - the template does not contain the required files.

How to configure Ubuntu Pro guest attachment

If Ubuntu Pro is enabled on a LXD host, guest instances can be automatically attached to the Pro subscription.

First, the Pro client on the host machine must be configured to allow guest attachment:

```
pro config set lxd_guest_attach=on
```

The allowed values are `on`, `off`, and `available`. If unset, this defaults to `off`.

> **Note**
>
> The Pro client must be updated to the latest version.

You can now launch an Ubuntu instance:

```
lxc launch ubuntu:24.04 pro-guest
```

The instance will automatically attach to the Pro subscription at start up.

Pro attachment can take some time. You can check the status using:

```
lxc exec pro-guest -- pro status
```

The `lxd_guest_attach` setting can be overridden by the *`ubuntu_pro.guest_attach`* (page 417) configuration option. For example, if `lxd_guest_attach` is set to `on` on the host, to prevent Pro attachment in the guest you can run:

```
lxc launch ubuntu:24.04 non-pro-guest -c ubuntu_pro.guest_attach=off
```

The `ubuntu_pro.guest_attach` configuration key has three options: `on`, `off`, and `available`.

All options for Pro guest attachment are described below.

	`on (host)`	`available (host)`	`off (host)`	`unset (host)`
`on (guest)`	auto-attach on start	auto-attach on start	guest attachment disabled	guest attachment disabled
`available (guest)`	attach on `pro auto-attach`	attach on `pro-auto-attach`	guest attachment disabled	guest attachment disabled
`off (guest)`	guest attachment disabled	guest attachment disabled	guest attachment disabled	guest attachment disabled
`unset (guest)`	auto-attach on start	attach on `pro-auto-attach`	guest attachment disabled	guest attachment disabled

How to work with instances:

How to access files in an instance

You can manage files inside an instance using the LXD client or the API without needing to access the instance through the network. Files can be individually edited or deleted, pushed from or pulled to the local machine. Alternatively, if you're using the LXD client, you can mount the instance's file system onto the local machine.

> **Note**
>
> The UI does not currently support accessing files in an instance.

For containers, these file operations always work and are handled directly by LXD. For virtual machines, the `lxd-agent` process must be running inside of the virtual machine for them to work.

Edit instance files

CLI

API

To edit an instance file from your local machine, enter the following command:

```
lxc file edit <instance_name>/<path_to_file>
```

For example, to edit the `/etc/hosts` file in the instance, enter the following command:

```
lxc file edit my-instance/etc/hosts
```

> **Note**
>
> The file must already exist on the instance. You cannot use the `edit` command to create a file on the instance.

There is no API endpoint that lets you edit files directly on an instance. Instead, you need to *pull the content of the file from the instance* (page 106), edit it, and then *push the modified content back to the instance* (page 107).

Delete files from the instance

CLI

API

To delete a file from your instance, enter the following command:

```
lxc file delete <instance_name>/<path_to_file>
```

Send the following DELETE request to delete a file from your instance:

```
lxc query --request DELETE /1.0/instances/<instance_name>/files?path=<path_to_
file>
```

See `DELETE /1.0/instances/{name}/files` for more information.

Pull files from the instance to the local machine

CLI

API

To pull a file from your instance to your local machine, enter the following command:

```
lxc file pull <instance_name>/<path_to_file> <local_file_path>
```

For example, to pull the `/etc/hosts` file to the current directory, enter the following command:

```
lxc file pull my-instance/etc/hosts .
```

Instead of pulling the instance file into a file on the local system, you can also pull it to stdout and pipe it to stdin of another command. This can be useful, for example, to check a log file:

```
lxc file pull my-instance/var/log/syslog - | less
```

To pull a directory with all contents, enter the following command:

```
lxc file pull -r <instance_name>/<path_to_directory> <local_location>
```

Send the following request to pull the contents of a file from your instance to your local machine:

```
lxc query --request GET /1.0/instances/<instance_name>/files?path=<path_to_file>
```

You can then write the contents to a local file, or pipe them to stdin of another command.

For example, to pull the contents of the `/etc/hosts` file and write them to a `my-instance-hosts` file in the current directory, enter the following command:

```
lxc query --request GET /1.0/instances/my-instance/files?path=/etc/hosts > my-
instance-hosts
```

To examine a log file, enter the following command:

```
lxc query --request GET /1.0/instances/<instance_name>/files?path=<file_path> |
less
```

To pull the contents of a directory, send the following request:

```
lxc query --request GET /1.0/instances/<instance_name>/files?path=<path_to_
directory>
```

This request returns a list of files in the directory, and you can then pull the contents of each file.

See `GET /1.0/instances/{name}/files` for more information.

Push files from the local machine to the instance

CLI

API

To push a file from your local machine to your instance, enter the following command:

```
lxc file push <local_file_path> <instance_name>/<path_to_file>
```

You can specify the file permissions by adding the `--gid`, `--uid`, and `--mode` flags.

To push a directory with all contents, enter the following command:

```
lxc file push -r <local_location> <instance_name>/<path_to_directory>
```

Send the following request to write content to a file on your instance:

```
lxc query --request POST /1.0/instances/<instance_name>/files?path=<path_to_file>
--data <content>
```

See `POST /1.0/instances/{name}/files` for more information.

To push content directly from a file, you must use a tool that can send raw data from a file, which *lxc query* (page 869) does not support. For example, with curl:

```
curl -X POST -H "Content-Type: application/octet-stream" --data-binary @<local_
file_path> \
--unix-socket /var/snap/lxd/common/lxd/unix.socket \
lxd/1.0/instances/<instance_name>/files?path=<path_to_file>
```

Mount a file system from the instance

CLI

API

You can mount an instance file system into a local path on your client.

To do so, make sure that you have `sshfs` installed. Then run the following command (note that if you're using the snap, the command requires root permissions):

```
lxc file mount <instance_name>/<path_to_directory> <local_location>
```

You can then access the files from your local machine.

Set up an SSH SFTP listener

Alternatively, you can set up an SSH SFTP listener. This method allows you to connect with any SFTP client and with a dedicated user name. Also, if you're using the snap, it does not require root permission.

To do so, first set up the listener by entering the following command:

```
lxc file mount <instance_name> [--listen <address>:<port>]
```

For example, to set up the listener on a random port on the local machine (for example, `127.0.0.1:45467`):

```
lxc file mount my-instance
```

If you want to access your instance files from outside your local network, you can pass a specific address and port:

```
lxc file mount my-instance --listen 192.0.2.50:2222
```

Caution

Be careful when doing this, because it exposes your instance remotely.

To set up the listener on a specific address and a random port:

```
lxc file mount my-instance --listen 192.0.2.50:0
```

The command prints out the assigned port and a user name and password for the connection.

Tip

You can specify a user name by passing the `--auth-user` flag.

Use this information to access the file system. For example, if you want to use `sshfs` to connect, enter the following command:

```
sshfs <user_name>@<address>:<path_to_directory> <local_location> -p <port>
```

For example:

```
sshfs xFn8ai8c@127.0.0.1:/home my-instance-files -p 35147
```

You can then access the file system of your instance at the specified location on the local machine.

Mounting a file system is not directly supported through the API, but requires additional processing logic on the client side.

How to access the console

You can access the instance console to log in to the instance and see log messages. The console is available at boot time already, so you can use it to see boot messages and, if necessary, debug startup issues of a container or VM.

CLI

API

UI

Use the *`lxc console`* (page 761) command to attach to instance consoles. To get an interactive console, enter the following command:

```
lxc console <instance_name>
```

To show new log messages (only for containers), pass the `--show-log` flag:

```
lxc console <instance_name> --show-log
```

You can also immediately attach to the console when you start your instance:

```
lxc start <instance_name> --console
lxc start <instance_name> --console=vga
```

Tip

To exit the console, enter `Ctrl+a q`.

To start an interactive console, send a POST request to the `console` endpoint:

```
lxc query --request POST /1.0/instances/<instance_name>/console --data '{
  "height": 24,
  "type": "console",
  "width": 80
}'
```

This query sets up two WebSockets that you can use for connection. One WebSocket is used for control, and the other transmits the actual console data.

See `POST /1.0/instances/{name}/console` for more information.

To access the WebSockets, you need the operation ID and the secrets for each socket. This information is available in the operation started by the query, for example:

```
{
  "class": "websocket",
  "created_at": "2024-01-31T10:11:48.135150288Z",
  "description": "Showing console",
  "err": "",
  "id": "<operation_ID>",
  "location": "none",
  "may_cancel": false,
  "metadata": {
    "fds": {
      "0": "<data_socket_secret>",
      "control": "<control_socket_secret>"
    }
  }
[...]
}
```

How to connect to the WebSockets depends on the tooling that you use (see `GET /1.0/operations/{id}/websocket` for general information). To quickly check whether the connection is successful and you can read from the socket, you can use a tool like `websocat`[64]:

```
websocat --text \
--ws-c-uri=ws://unix.socket/1.0/operations/<operation_ID>/websocket?secret=<data_
socket_secret> \
- ws-c:unix:/var/snap/lxd/common/lxd/unix.socket
```

Alternatively, if you just want to retrieve new log messages from the console instead of connecting through a WebSocket, you can send a GET request to the `console` endpoint:

```
lxc query --request GET /1.0/instances/<instance_name>/console
```

See `GET /1.0/instances/{name}/console` for more information. Note that this operation is supported only for containers, not for VMs.

Navigate to the instance detail page and switch to the *Console* tab to view the console.

64 https://github.com/vi/websocat

Access the graphical console (for virtual machines)

On virtual machines, log on to the console to get graphical output. Using the console you can, for example, install an operating system using a graphical interface or run a desktop environment.

An additional advantage is that the console is available even if the `lxd-agent` process is not running. This means that you can access the VM through the console before the `lxd-agent` starts up, and also if the `lxd-agent` is not available at all.

CLI

API

UI

To start the VGA console with graphical output for your VM, you must install a SPICE client (for example, `virt-viewer` or `spice-gtk-client`). Then enter the following command:

```
lxc console <vm_name> --type vga
```

To start the VGA console with graphical output for your VM, send a POST request to the `console` endpoint:

```
lxc query --request POST /1.0/instances/<instance_name>/console --data '{
  "height": 0,
  "type": "vga",
  "width": 0
}'
```

See `POST /1.0/instances/{name}/console` for more information.

Navigate to the instance detail page and switch to the *Console* tab to view the console.

For virtual machines, you can switch between the graphic console and the text console.

How to run commands in an instance

LXD allows to run commands inside an instance using the LXD client or the API, without needing to access the instance through the network.

For containers, this always works and is handled directly by LXD. For virtual machines, the `lxd-agent` process must be running inside of the virtual machine for this to work.

> **Note**
>
> The UI does not currently support sending commands to an instance. However, it provides a terminal that gives you *shell access to your instance* (page 115).

Run commands inside your instance

CLI

API

To run a single command from the terminal of the host machine, use the *lxc exec* (page 763) command:

```
lxc exec <instance_name> -- <command>
```

For example, enter the following command to update the package list on your container:

```
lxc exec my-instance -- apt-get update
```

Send a POST request to the instance's `exec` endpoint to run a single command from the terminal of the host machine:

```
lxc query --request POST /1.0/instances/<instance_name>/exec --data '{
  "command": [ "<command>" ]
}'
```

For example, enter the following command to update the package list on your container:

```
lxc query --request POST /1.0/instances/my-instance/exec --data '{
  "command": [ "apt-get", "update" ]
}'
```

See `POST /1.0/instances/{name}/exec` for more information.

Execution mode

LXD can execute commands either interactively or non-interactively.

CLI

API

In interactive mode, a pseudo-terminal device (PTS) is used to handle input (stdin) and output (stdout, stderr). This mode is automatically selected by the CLI if connected to a terminal emulator (and not run from a script). To force interactive mode, add either `--force-interactive` or `--mode interactive` to the command.

In non-interactive mode, pipes are allocated instead (one for each of stdin, stdout and stderr). This method allows running a command and properly getting separate stdin, stdout and stderr as required by many scripts. To force non-interactive mode, add either `--force-noninteractive` or `--mode non-interactive` to the command.

In both modes, the operation creates a control socket that can be used for out-of-band communication with LXD. You can send signals and window sizing information through this socket.

Interactive mode

In interactive mode, the operation creates an additional single bi-directional WebSocket. To force interactive mode, add `"interactive": true` and `"wait-for-websocket": true` to the request data. For example:

```
lxc query --request POST /1.0/instances/my-instance/exec --data '{
  "command": [ "/bin/bash" ],
  "interactive": true,
  "wait-for-websocket": true
}'
```

Non-interactive mode

In non-interactive mode, the operation creates three additional WebSockets: one each for stdin, stdout, and stderr. To force non-interactive mode, add `"interactive": false` to the request data.

When running a command in non-interactive mode, you can instruct LXD to record the output of the command. To do so, add `"record-output": true` to the request data. You can then send a request to the `exec-output` endpoint to retrieve the list of files that contain command output:

```
lxc query --request GET /1.0/instances/<instance_name>/logs/exec-output
```

To display the output of one of the files, send a request to one of the files:

```
lxc query --request GET /1.0/instances/<instance_name>/logs/exec-output/
<record-output-file>
```

When you don't need the command output anymore, you can delete it:

```
lxc query --request DELETE /1.0/instances/<instance_name>/logs/exec-output/
<record-output-file>
```

See `GET /1.0/instances/{name}/logs/exec-output`, `GET /1.0/instances/{name}/logs/exec-output/{filename}`, and `DELETE /1.0/instances/{name}/logs/exec-output/{filename}` for more information.

User, groups and working directory

LXD has a policy not to read data from within the instances or trust anything that can be found in the instance. Therefore, LXD does not parse files like `/etc/passwd`, `/etc/group` or `/etc/nsswitch.conf` to handle user and group resolution.

As a result, LXD doesn't know the home directory for the user or the supplementary groups the user is in.

By default, LXD runs commands as `root` (UID 0) with the default group (GID 0) and the working directory set to `/root`. You can override the user, group and working directory by specifying absolute values.

CLI

API

You can override the default settings by adding the following flags to the *lxc exec* (page 763) command:

- `--user` - the user ID for running the command
- `--group` - the group ID for running the command
- `--cwd` - the directory in which the command should run

You can override the default settings by adding the following fields to the request data:

- `"user": <user_ID>` - the user ID for running the command
- `"group": <group_ID>` - the group ID for running the command
- `"cwd": "<directory>"` - the directory in which the command should run

Environment

You can pass environment variables to an exec session in the following two ways:

Set environment variables as instance options

CLI

API

UI

To set the `<ENVVAR>` environment variable to `<value>` in the instance, set the `environment.<ENVVAR>` instance option (see *environment.** (page 418)):

```
lxc config set <instance_name> environment.<ENVVAR>=<value>
```

To set the `<ENVVAR>` environment variable to `<value>` in the instance, set the `environment.<ENVVAR>` instance option (see *environment.** (page 418)):

```
lxc query --request PATCH /1.0/instances/<instance_name> --data '{
  "config": {
    "environment.<ENVVAR>": "<value>"
  }
}'
```

To set the `<ENVVAR>` environment variable to `<value>` in the instance, go to the instance detail page, switch to the *Configuration* tab and select *YAML configuration*. Then click *Edit instance*.

Add the `environment.<ENVVAR>` configuration under the `config` section. For example:

```
config:
  environment.<ENVVAR>: "<value>"
```

Click *Save changes*.

Pass environment variables to the exec command

CLI

API

To pass an environment variable to the exec command, use the `--env` flag. For example:

```
lxc exec <instance_name> --env <ENVVAR>=<value> -- <command>
```

To pass an environment variable to the exec command, add an `environment` field to the request data. For example:

```
lxc query --request POST /1.0/instances/<instance_name>/exec --data '{
  "command": [ "<command>" ],
  "environment": {
    "<ENVVAR>": "<value>"
  }
}'
```

In addition, LXD sets the following default values (unless they are passed in one of the ways described above):

Variable name	Condition	Value
PATH	-	Concatenation of: • /usr/local/sbin • /usr/local/bin • /usr/sbin • /usr/bin • /sbin • /bin • /snap (if applicable) • /etc/NIXOS (if applicable)
LANG	-	C.UTF-8
HOME	running as root (UID 0)	/root
USER	running as root (UID 0)	root

Get shell access to your instance

If you want to run commands directly in your instance, run a shell command inside it.

CLI

API

UI

Enter the following command (assuming that the /bin/bash command exists in your instance):

```
lxc exec <instance_name> -- /bin/bash
```

Enter the following command (assuming that the /bin/bash command exists in your instance):

```
lxc query --request POST /1.0/instances/<instance_name>/exec --data '{
  "command": [ "/bin/bash" ]
}'
```

Navigate to the instance detail page and switch to the *Terminal* tab to access the shell.

By default, you are logged in as the root user. If you want to log in as a different user, enter the following command:

CLI

API

UI

```
lxc exec <instance_name> -- su --login <user_name>
```

To exit the instance shell, enter exit or press Ctrl+d.

```
lxc query --request POST /1.0/instances/<instance_name>/exec --data '{
  "command": [ "su", "--login", "<user_name>" ]
}'
```

```
su --login <user_name>
```

To exit the user shell and go back to the root shell, enter `exit` or press `Ctrl+d`.

> **Note**
>
> Depending on the operating system that you run in your instance, you might need to create a user first.

How to use `cloud-init`

`cloud-init`[65] is a tool for automatically initializing and customizing an instance of a Linux distribution.

By adding `cloud-init` configuration to your instance, you can instruct `cloud-init` to execute specific actions at the first start of an instance. Possible actions include, for example:

- Updating and installing packages
- Applying certain configurations
- Adding users
- Enabling services
- Running commands or scripts
- Automatically growing the file system of a VM to the size (quota) of the disk

See the Cloud-init documentation[66] for detailed information.

> **Note**
>
> The `cloud-init` actions are run only once on the first start of the instance. Rebooting the instance does not re-trigger the actions.

`cloud-init` support in images

To use `cloud-init`, you must base your instance on an image that has `cloud-init` installed:

- All images from the `ubuntu` and `ubuntu-daily` *image servers* (page 391) have `cloud-init` support. However, images for Ubuntu releases prior to 20.04 LTS require special handling to integrate properly with `cloud-init`, so that `lxc exec` works correctly with virtual machines that use those images. Refer to *VM `cloud-init`* (page 479).

[65] https://cloud-init.io/
[66] https://cloudinit.readthedocs.io/en/latest/index.html#index

- Images from the images remote[67] have cloud-init-enabled variants, which are usually bigger in size than the default variant. The cloud variants use the /cloud suffix, for example, images:alpine/edge/cloud.

Configuration options

LXD supports two different sets of configuration options for configuring cloud-init: cloud-init.* and user.*. Which of these sets you must use depends on the cloud-init support in the image that you use. As a rule of thumb, newer images support the cloud-init.* configuration options, while older images support user.*. However, there might be exceptions to that rule.

The following configuration options are supported:

- cloud-init.vendor-data or user.vendor-data (see Vendor-data[68])
- cloud-init.user-data or user.user-data (see User-data formats[69])
- cloud-init.network-config or user.network-config (see Network configuration[70])

For more information about the configuration options, see the *cloud-init instance options* (page 419), and the documentation for the LXD data source[71] in the cloud-init documentation.

> **Note**
>
> Ubuntu 20.04 and earlier have recent versions of the cloud-init package but support for the modern cloud-init.* configuration options is disabled in those series. As such, when using such old instances, remember to use the user.* configuration options instead.

Vendor data and user data

Both vendor-data and user-data are used to provide cloud configuration data[72] to cloud-init.

The main idea is that vendor-data is used for the general default configuration, while user-data is used for instance-specific configuration. This means that you should specify vendor-data in a profile and user-data in the instance configuration. LXD does not enforce this method, but allows using both vendor-data and user-data in profiles and in the instance configuration.

If both vendor-data and user-data are supplied for an instance, cloud-init merges the two configurations. However, if you use the same keys in both configurations, merging might not be possible. In this case, configure how cloud-init should merge the provided data. See Merging user-data sections[73] for instructions.

[67] https://images.lxd.canonical.com/
[68] https://cloudinit.readthedocs.io/en/latest/explanation/vendordata.html#vendor-data
[69] https://cloudinit.readthedocs.io/en/latest/explanation/format.html#user-data-formats
[70] https://cloudinit.readthedocs.io/en/latest/reference/network-config.html#network-config
[71] https://cloudinit.readthedocs.io/en/latest/reference/datasources/lxd.html#datasource-lxd
[72] https://cloudinit.readthedocs.io/en/latest/explanation/format.html#cloud-config-data
[73] https://cloudinit.readthedocs.io/en/latest/reference/merging.html#merging-user-data

How to configure `cloud-init`

To configure `cloud-init` for an instance, add the corresponding configuration options to a *profile* (page 97) that the instance uses or directly to the *instance configuration* (page 84).

When configuring `cloud-init` directly for an instance, keep in mind that `cloud-init` runs only on instance start. This means any changes to `cloud-init` configuration only take effect after the next instance start. To ensure `cloud-init` configurations are applied on every boot, LXD changes the instance ID whenever relevant `cloud-init` configuration keys are modified. This triggers `cloud-init` to fetch and apply the updated data from LXD as if it were the instance's first boot. For more information, see the `cloud-init` docs regarding First boot determination[74].

To add your configuration:

CLI

API

UI

Write the configuration to a file and pass that file to the `lxc config` command. For example, create `cloud-init.yml` with the following content:

```
#cloud-config
package_upgrade: true
packages:
  - package1
  - package2
```

Then run the following command:

```
lxc config set <instance_name> cloud-init.user-data - < cloud-init.yml
```

Provide the `cloud-init` configuration as a string with escaped newline characters.

For example:

```
lxc query --request PATCH /1.0/instances/<instance_name> --data '{
  "config": {
    "cloud-init.user-data": "#cloud-config\npackage_upgrade: true\npackages:\n  -
package1\n  - package2"
  }
}'
```

Alternatively, to avoid mistakes, write the configuration to a file and include that in your request. For example, create `cloud-init.txt` with the following content:

```
#cloud-config
package_upgrade: true
packages:
  - package1
  - package2
```

Then send the following request:

[74] https://cloudinit.readthedocs.io/en/latest/explanation/first_boot.html#first-boot-determination

```
lxc query --request PATCH /1.0/instances/<instance_name> --data '{
"config": {
  "cloud-init.user-data": "'"$(awk -v ORS='\\n' '1' cloud-init.txt)"'"
  }
}'
```

Go to the *Configuration* tab of the instance detail page and select *Advanced > Cloud init*. Then click *Edit instance* and override the configuration for one or more of the `cloud-init` configuration options.

YAML format for `cloud-init` configuration

The `cloud-init` options require YAML's literal style format[75]. You use a pipe symbol (|) to indicate that all indented text after the pipe should be passed to `cloud-init` as a single string, with new lines and indentation preserved.

The `vendor-data` and `user-data` options usually start with `#cloud-config`. But `cloud-init` has an array of configuration formats[76] available.

For example:

```
config:
  cloud-init.user-data: |
    #cloud-config
    package_upgrade: true
    packages:
      - package1
      - package2
```

```
config:
  cloud-init.user-data: |
    #!/usr/bin/bash
    echo hello | tee -a /tmp/example.txt
```

Tip

See How to validate user data[77] for information on how to check whether the syntax is correct.

How to check the `cloud-init` status

`cloud-init` runs automatically on the first start of an instance. Depending on the configured actions, it might take a while until it finishes.

To check the `cloud-init` status, log on to the instance and enter the following command:

```
cloud-init status
```

[75] https://yaml.org/spec/1.2.2/#812-literal-style
[76] https://docs.cloud-init.io/en/latest/explanation/format.html#configuration-types
[77] https://cloudinit.readthedocs.io/en/latest/howto/debug_user_data.html#check-user-data-cloud-config

If the result is `status: running`, `cloud-init` is still working. If the result is `status: done`, it has finished.

Alternatively, use the `--wait` flag to be notified only when `cloud-init` is finished:

```
root@instance:~# cloud-init status --wait
.....................................status: done
```

How to specify user or vendor data

The `user-data` and `vendor-data` configuration can be used to, for example, upgrade or install packages, add users, or run commands.

The provided values must have a first line that indicates what type of user data format[78] is being passed to `cloud-init`. For activities like upgrading packages or setting up a user, `#cloud-config` is the data format to use.

The configuration data is stored in the following files in the instance's root file system:

- `/var/lib/cloud/instance/cloud-config.txt`
- `/var/lib/cloud/instance/user-data.txt`

Examples

See the following sections for the user data (or vendor data) configuration for different example use cases.

You can find more advanced examples[79] in the `cloud-init` documentation.

Upgrade packages

To trigger a package upgrade from the repositories for the instance right after the instance is created, use the `package_upgrade` key:

```yaml
config:
  cloud-init.user-data: |
    #cloud-config
    package_upgrade: true
```

Install packages

To install specific packages when the instance is set up, use the `packages` key and specify the package names as a list:

```yaml
config:
  cloud-init.user-data: |
    #cloud-config
    packages:
      - git
      - openssh-server
```

[78] https://cloudinit.readthedocs.io/en/latest/explanation/format.html#user-data-formats
[79] https://cloudinit.readthedocs.io/en/latest/reference/examples.html#yaml-examples

Set the time zone

To set the time zone for the instance on instance creation, use the `timezone` key:

```
config:
  cloud-init.user-data: |
    #cloud-config
    timezone: Europe/Rome
```

Run commands

To run a command (such as writing a marker file), use the `runcmd` key and specify the commands as a list:

```
config:
  cloud-init.user-data: |
    #cloud-config
    runcmd:
      - [touch, /run/cloud.init.ran]
```

Add a user account

To add a user account, use the `users` key. See the Including users and groups[80] example in the `cloud-init` documentation for details about default users and which keys are supported.

```
config:
  cloud-init.user-data: |
    #cloud-config
    users:
      - name: documentation_example
```

How to specify network configuration data

By default, `cloud-init` configures a DHCP client on an instance's `eth0` interface. You can define your own network configuration using the `network-config` option to override the default configuration (this is due to how the template is structured).

`cloud-init` then renders the relevant network configuration on the system using either `ifupdown` or `netplan`, depending on the Ubuntu release.

The configuration data is stored in the following files in the instance's root file system:

- `/var/lib/cloud/seed/nocloud-net/network-config`
- `/etc/network/interfaces.d/50-cloud-init.cfg` (if using `ifupdown`)
- `/etc/netplan/50-cloud-init.yaml` (if using `netplan`)

80 https://cloudinit.readthedocs.io/en/latest/reference/examples.html#including-users-and-groups

Example

To configure a specific network interface with a static IPv4 address and also use a custom name server, use the following configuration:

```
config:
  cloud-init.network-config: |
    version: 2
    ethernets:
      eth1:
        addresses:
          - 10.10.101.20/24
        gateway4: 10.10.101.1
        nameservers:
          addresses:
            - 10.10.10.254
```

How to inject SSH keys into instances

To inject SSH keys into LXD instances for an arbitrary user, use the configuration key `cloud-init.ssh-keys.<keyName>`.

Use the format `<user>:<key>` for its value, where `<user>` is a Linux username and `<key>` can be either a pure SSH public key or an import ID for a key hosted elsewhere. For example, `root:gh:githubUser` and `myUser:ssh-keyAlg publicKeyHash` are valid values. To prevent a particular SSH key from being inherited from a profile by an instance, edit the instance configuration by setting the `cloud-init.ssh-keys.<keyName>` key that references the target SSH key to `none`, and the key will not be injected.

The contents of the `cloud-init.ssh-keys.<keyName>` keys are merged into both *cloud-init.vendor-data* (page 420) and *cloud-init.user-data* (page 420) before being passed to the guest, following the `cloud-config` specification. (See the cloud-init documentation[81] for details.) Therefore, keys defined via `cloud-init.ssh-keys.<keyName>` cannot be applied if LXD cannot parse the existing `cloud-init.vendor-data` and `cloud-init.user-data` for that instance. This might occur if those keys are not in YAML format or contain invalid YAML. Other configuration formats are not yet supported.

You can define SSH keys via `cloud-init.vendor-data` or `cloud-init.user-data` directly. Keys defined using `cloud-init.ssh-keys.<keyName>` do not conflict with those defined in either of those settings. For details on defining SSH keys with `cloud-config`, see the cloud-init documentation for SSH configuration[82]. Changing a `cloud-init.*` key does not remove previously applied keys.

Since `cloud-init` only runs on instance start, updates to `cloud-init.*` keys on a running instance only take effect after restart.

[81] https://cloudinit.readthedocs.io/en/latest/explanation/about-cloud-config.html#about-cloud-config
[82] https://cloudinit.readthedocs.io/en/latest/reference/yaml_examples/ssh.html#cce-ssh

Examples

The following command injects `someuser`'s key from Launchpad into the newly created `container`:

```
lxc launch ubuntu:24.04 container -c cloud-init.ssh-keys.mykey=root:lp:someuser
```

The example profile configuration below defines a key to be injected on an instance. The injected key enables the owner of the private key to SSH into the instance as a user named `user`:

```
config:
  cloud-init.vendor-data: |
    users:
      - name: user
        ssh_authorized_keys: ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIJFDWcYmMrCZdk9JI29bAiHKD90oEUr8tqK5Vvo08Vcj
```

How to add a routed NIC device to a virtual machine

When adding a *routed NIC device* (page 472) to an instance, you must configure the instance to use the link-local gateway IPs as default routes. For containers, this is configured for you automatically. For virtual machines, the gateways must be configured manually or via a mechanism like `cloud-init`.

To configure the gateways with `cloud-init`, firstly initialize an instance:

CLI

API

UI

```
lxc init ubuntu:24.04 my-vm --vm
```

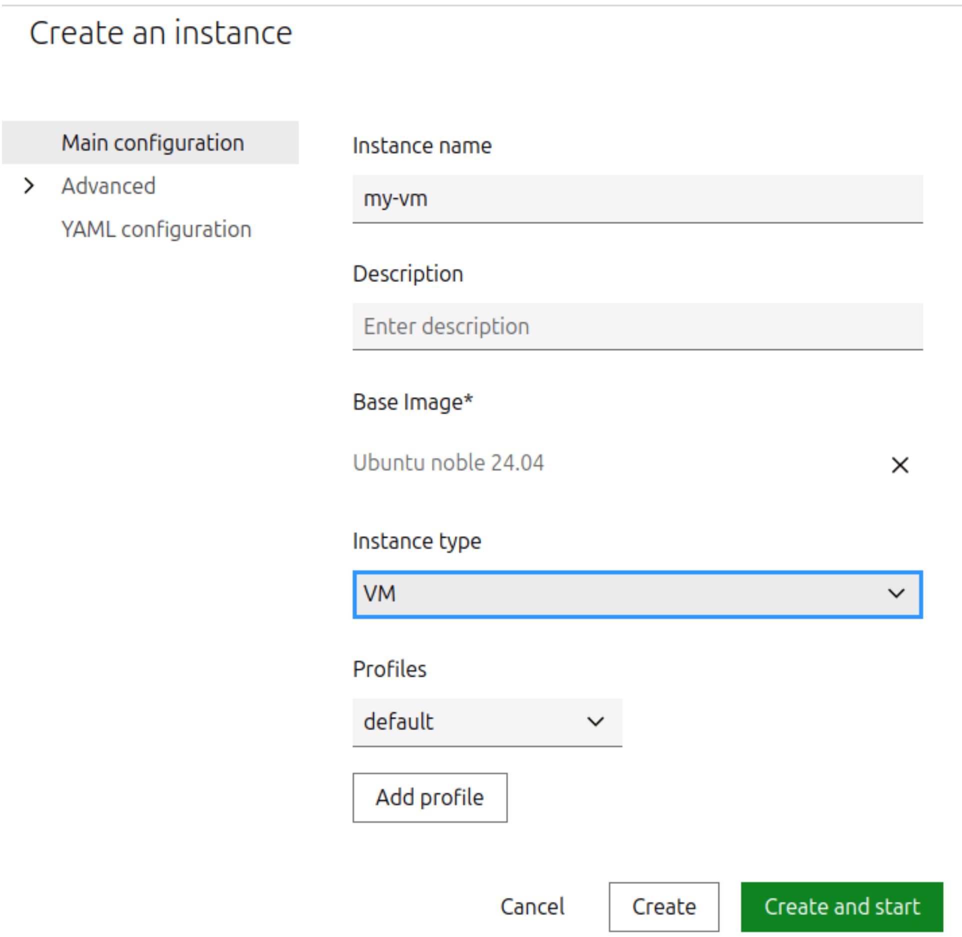
```
lxc query --request POST /1.0/instances --data '{
  "name": "my-vm",
  "source": {
    "alias": "24.04",
    "protocol": "simplestreams",
    "server": "https://cloud-images.ubuntu.com/releases/",
    "type": "image"
  },
  "type": "virtual-machine"
}'
```

Then add the routed NIC device:

CLI

API

UI


Create an instance
Main configuration
Advanced
YAML configuration
Instance name
my-vm
Description
Enter description
Base Image*
Ubuntu noble 24.04
Instance type
VM
Profiles
default
Add profile
Cancel
Create
Create and start

```
lxc config device add my-vm eth0 nic nictype=routed parent=my-parent ipv4.
address=192.0.2.2 ipv6.address=2001:db8::2
```

```
lxc query --request PATCH /1.0/instances/my-vm --data '{
  "devices": {
    "eth0": {
      "ipv4.address": "192.0.2.2",
      "ipv6.address": "2001:db8::2",
      "nictype": "routed",
      "parent": "my-parent",
      "type": "nic"
    }
  }
}'
```

You cannot add a routed NIC device through the UI directly. Therefore, go to the instance detail page, switch to the *Configuration* tab and select *YAML configuration*. Then click *Edit instance* and add the routed NIC device to the `devices` section. For example:

```
devices:
  eth0:
    ipv4.address: 192.0.2.2
    ipv6.address: 2001:db8::2
    nictype: routed
    parent: my-parent
    type: nic
```

In this configuration, `my-parent` is your parent network, and the IPv4 and IPv6 addresses are within the subnet of the parent.

Next we will add some `netplan` configuration to the instance using the `cloud-init.network-config` configuration key:

CLI

API

UI

```
cat <<EOF | lxc config set my-vm cloud-init.network-config -
network:
  version: 2
  ethernets:
    enp5s0:
      routes:
      - to: default
        via: 169.254.0.1
        on-link: true
      - to: default
        via: fe80::1
        on-link: true
      addresses:
```

(continues on next page)

```
      - 192.0.2.2/32
      - 2001:db8::2/128
EOF
```

```
cat > cloud-init.txt <<EOF
network:
  version: 2
  ethernets:
    enp5s0:
      routes:
      - to: default
        via: 169.254.0.1
        on-link: true
      - to: default
        via: fe80::1
        on-link: true
      addresses:
      - 192.0.2.2/32
      - 2001:db8::2/128
EOF

lxc query --request PATCH /1.0/instances/my-vm --data '{
  "config": {
    "cloud-init.network-config": "'"$(awk -v ORS='\\n' '1' cloud-init.txt)"'"
  }
}'
```

On the instance detail page, switch to the *Advanced* > *Cloud-init* tab and click *Edit instance*.

Click the *Create override* icon for the *Network config* and enter the following configuration:

```
network:
  version: 2
  ethernets:
    enp5s0:
      routes:
      - to: default
        via: 169.254.0.1
        on-link: true
      - to: default
        via: fe80::1
        on-link: true
      addresses:
      - 192.0.2.2/32
      - 2001:db8::2/128
```

This `netplan` configuration adds the *static link-local next-hop addresses* (page 472) (`169.254.0.1` and `fe80::1`) that are required. For each of these routes we set `on-link` to `true`, which specifies that the route is directly connected to the interface. We also add the addresses that we configured in our routed NIC device. For more information on `netplan`, see their documentation[83].

> **Note**
>
> This `netplan` configuration does not include a name server. To enable DNS within the instance, you must set a valid DNS IP address. If there is a `lxdbr0` network on the host, the name server can be set to that IP instead.

Before you start your instance, make sure that you have *configured the parent network* (page 473) to enable proxy ARP/NDP.

Then start your instance:

CLI

API

UI

```
lxc start my-vm
```

```
lxc query --request PUT /1.0/instances/my-vm/state --data '{"action": "start"}'
```

Go to the instance list or the respective instance and click the *Start* button ().

How to export and move instances:

How to back up instances

There are different ways of backing up your instances:

- *Use snapshots for instance backup* (page 128)
- *Use export files for instance backup* (page 132)
- *Copy an instance to a backup server* (page 135)

Which method to choose depends both on your use case and on the storage driver you use.

In general, snapshots are quick and space efficient (depending on the storage driver), but they are stored in the same storage pool as the instance and therefore not too reliable. Export files can be stored on different disks and are therefore more reliable. They can also be used to restore the instance into a different storage pool. If you have a separate, network-connected LXD server available, regularly copying instances to this other server gives high reliability as well, and this method can also be used to back up snapshots of the instance.

> **Note**
>
> Custom storage volumes might be attached to an instance, but they are not part of the instance. Therefore, the content of a custom storage volume is not stored when you back up your instance. You must back up the data of your storage volume separately. See *How to back up custom storage volumes* (page 200) for instructions.

[83] https://netplan.readthedocs.io/en/latest/

Use snapshots for instance backup

You can save your instance at a point in time by creating an instance snapshot, which makes it easy to restore the instance to a previous state.

Instance snapshots are stored in the same storage pool as the instance volume itself.

Most storage drivers support optimized snapshot creation (see *Feature comparison* (page 571)). For these drivers, creating snapshots is both quick and space-efficient. For the `dir` driver, snapshot functionality is available but not very efficient. For the `lvm` driver, snapshot creation is quick, but restoring snapshots is efficient only when using thin-pool mode.

Create a snapshot

CLI

API

UI

Use the following command to create a snapshot of an instance:

```
lxc snapshot <instance_name> [<snapshot name>]
```

The snapshot name is optional. If you don't specify one, the name follows the naming pattern defined in `snapshots.pattern`.

Add the `--reuse` flag in combination with a snapshot name to replace an existing snapshot.

By default, snapshots are kept forever, unless the `snapshots.expiry` configuration option is set. To retain a specific snapshot even if a general expiry time is set, use the `--no-expiry` flag.

For virtual machines, you can add the `--stateful` flag to capture not only the data included in the instance volume but also the running state of the instance. Note that this feature is not fully supported for containers because of CRIU limitations.

To create a snapshot of an instance, send a POST request to the `snapshots` endpoint:

```
lxc query --request POST /1.0/instances/<instance_name>/snapshots --data '{"name":
"<snapshot_name>"}'
```

The snapshot name is optional. If you set it to an empty string, the name follows the naming pattern defined in *`snapshots.pattern`* (page 442).

By default, snapshots are kept forever, unless the *`snapshots.expiry`* (page 441) configuration option is set. To set an expiration date, add the`expires_at` field to the request data. To retain a specific snapshot even if a general expiry time is set, set the `expires_at` field to `"0001-01-01T00:00:00Z"`.

If you want to replace an existing snapshot, *delete it* (page 129) first and then create another snapshot with the same name.

For virtual machines, you can add `"stateful": true` to the request data to capture not only the data included in the instance volume but also the running state of the instance. Note that this feature is not fully supported for containers because of CRIU limitations.

See `POST /1.0/instances/{name}/snapshots` for more information.

To create a snapshot of an instance, go to the instance detail page and switch to the *Snapshots* tab. Click *Create snapshot* to open the dialog to create a snapshot.

The snapshot name is optional. If you don't specify one, the name follows the naming pattern defined in `snapshots.pattern` (page 442). You can check and update this option by switching to the *Configuration* tab and selecting *Advanced* > *Snapshots*, or simply by clicking *See configuration*.

By default, snapshots are kept forever, unless you specify an expiry date and time, or the `snapshots.expiry` (page 441) configuration option is set for the instance.

For virtual machines, you can choose to create a stateful snapshot to capture not only the data included in the instance volume but also the running state of the instance. Note that this feature requires `migration.stateful` (page 429) to be enabled.

View, edit or delete snapshots

CLI

API

UI

Use the following command to display the snapshots for an instance:

```
lxc info <instance_name>
```

You can view or modify snapshots in a similar way to instances, by referring to the snapshot with `<instance_name>/<snapshot_name>`.

To show configuration information about a snapshot, use the following command:

```
lxc config show <instance_name>/<snapshot_name>
```

To change the expiry date of a snapshot, use the following command:

```
lxc config edit <instance_name>/<snapshot_name>
```

> **Note**
>
> In general, snapshots cannot be edited, because they preserve the state of the instance. The only exception is the expiry date. Other changes to the configuration are silently ignored.

To delete a snapshot, use the following command:

```
lxc delete <instance_name>/<snapshot_name>
```

To retrieve the snapshots for an instance, send a GET request to the `snapshots` endpoint:

```
lxc query --request GET /1.0/instances/<instance_name>/snapshots
```

To show configuration information about a snapshot, send the following request:

```
lxc query --request GET /1.0/instances/<instance_name>/snapshots/<snapshot_name>
```

To change the expiry date of a snapshot, send a PATCH request:

```
lxc query --request PATCH /1.0/instances/<instance_name>/snapshots/<snapshot_name>
--data '{
  "expires_at": "2029-03-23T17:38:37.753398689-04:00"
}'
```

> **Note**
>
> In general, snapshots cannot be modified, because they preserve the state of the instance. The only exception is the expiry date. Other changes to the configuration are silently ignored.

To delete a snapshot, send a DELETE request:

```
lxc query --request DELETE /1.0/instances/<instance_name>/snapshots/<snapshot_
name>
```

See `GET /1.0/instances/{name}/snapshots`, `GET /1.0/instances/{name}/snapshots/{snapshot}`, `PATCH /1.0/instances/{name}/snapshots/{snapshot}`, and `DELETE /1.0/instances/{name}/snapshots/{snapshot}` for more information.

To see all snapshots for an instance, go to the instance detail page and switch to the *Snapshots* tab.

From the snapshot list, you can choose to edit the name or expiry date of a specific snapshot, create an image based on the snapshot, restore it to the instance, or delete it.

Schedule instance snapshots

You can configure an instance to automatically create snapshots at specific times (at most once every minute). To do so, set the *snapshots.schedule* (page 442) instance option.

For example, to configure daily snapshots:

CLI

API

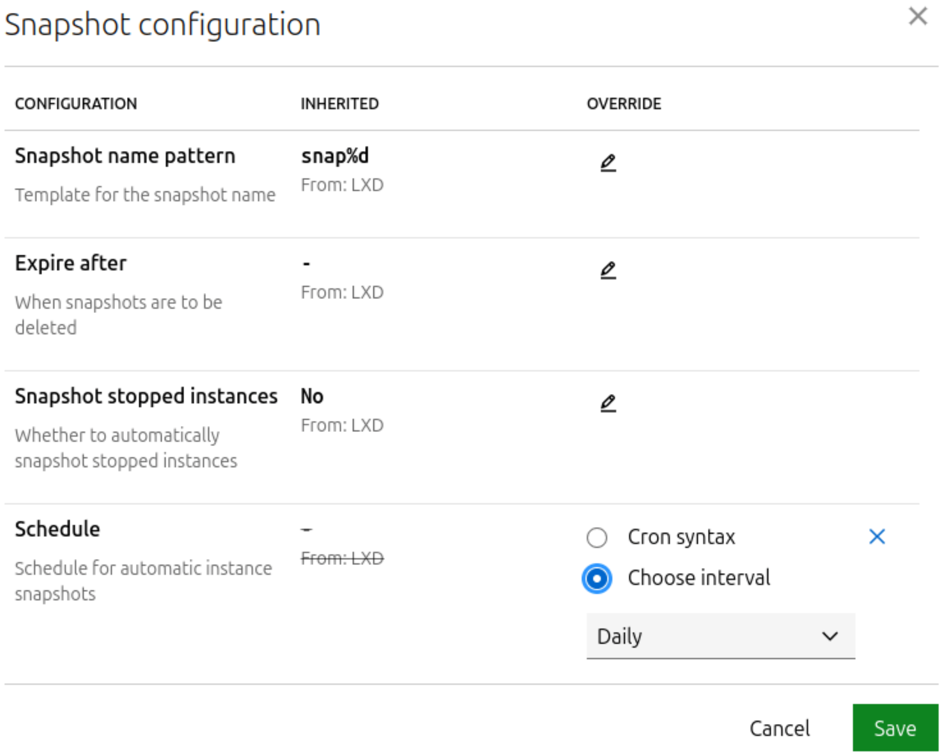
UI

```
lxc config set <instance_name> snapshots.schedule @daily
```

```
lxc query --request PATCH /1.0/instances/<instance_name> --data '{
  "config": {
    "snapshots.schedule": "@daily"
  }
}'
```

To configure taking a snapshot every day at 6 am:

CLI



API

UI

```
lxc config set <instance_name> snapshots.schedule "0 6 * * *"
```

```
lxc query --request PATCH /1.0/instances/<instance_name> --data '{
  "config": {
    "snapshots.schedule": "0 6 * * *"
  }
}'
```

When scheduling regular snapshots, consider setting an automatic expiry (*snapshots.expiry* (page 441)) and a naming pattern for snapshots (*snapshots.pattern* (page 442)). You should also configure whether you want to take snapshots of instances that are not running (*snapshots.schedule.stopped* (page 442)).

Restore an instance snapshot

You can restore an instance to any of its snapshots.

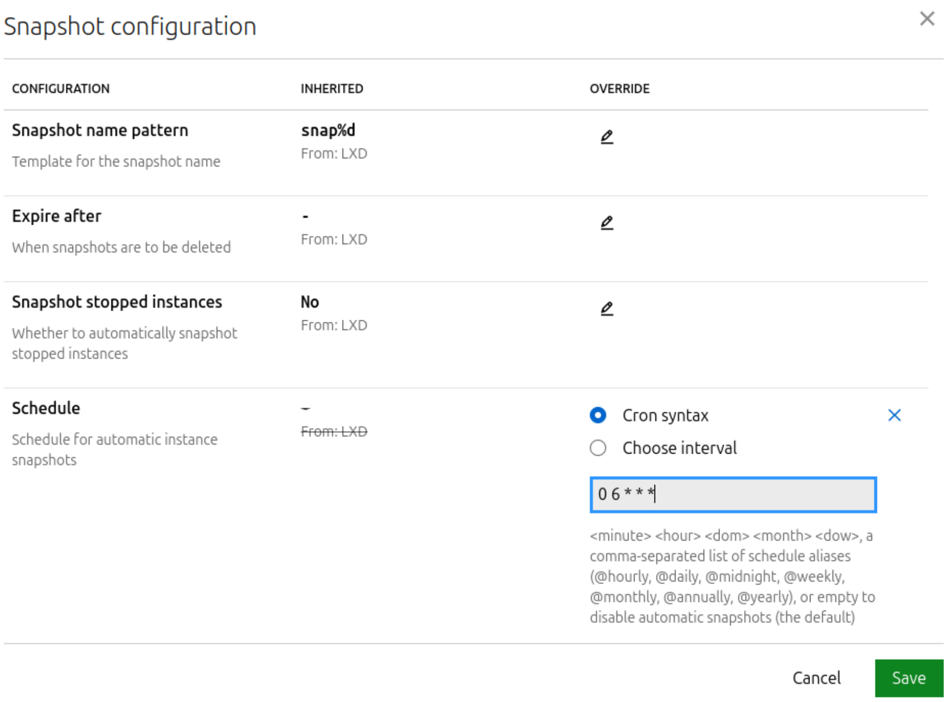
CLI

API

UI

To restore an instance to a snapshot, use the following command:

```
lxc restore <instance_name> <snapshot_name>
```



If the snapshot is stateful (which means that it contains information about the running state of the instance), you can add the `--stateful` flag to restore the state.

To restore an instance to a snapshot, send a PUT request to the instance:

```
lxc query --request PUT /1.0/instances/<instance_name> --data '{
  "restore": "<instance_name>/<snapshot_name>"
}'
```

If the snapshot is stateful (which means that it contains information about the running state of the instance), you can add `"stateful": true` to the request data:

```
lxc query --request PUT /1.0/instances/<instance_name> --data '{
  "restore": "<instance_name>/<snapshot_name>",
  "stateful": true
}'
```

See `PUT /1.0/instances/{name}` for more information.

To restore an instance to a snapshot, click the *Restore snapshot* button () next to the snapshot that you want to restore.

If the snapshot is stateful (which means that it contains information about the running state of the instance), select *Restore the instance state* if you want to restore the state.

Use export files for instance backup

You can export the full content of your instance to a standalone file that can be stored at any location. For highest reliability, store the backup file on a different file system to ensure that it does not get lost or corrupted.

Note

The UI does not currently support exporting and importing instances.

Export an instance

CLI

API

Use the following command to export an instance to a compressed file (for example, `/path/to/my-instance.tgz`):

```
lxc export <instance_name> [<file_path>]
```

If you do not specify a file path, the export file is saved as `<instance_name>.<extension>` in the working directory (for example, `my-container.tar.gz`).

Warning

If the output file (`<instance_name>.<extension>` or the specified file path) already exists, the command overwrites the existing file without warning.

You can add any of the following flags to the command:

`--compression`
: By default, the output file uses `gzip` compression. You can specify a different compression algorithm (for example, `bzip2`) or turn off compression with `--compression=none`.

`--optimized-storage`
: If your storage pool uses the `btrfs` or the `zfs` driver, add the `--optimized-storage` flag to store the data as a driver-specific binary blob instead of an archive of individual files. In this case, the export file can only be used with pools that use the same storage driver.

  Exporting a volume in optimized mode is usually quicker than exporting the individual files. Snapshots are exported as differences from the main volume, which decreases their size (quota) and makes them easily accessible.

`--export-version`
: If you intend to import the backup to an older version of LXD, set the version to `1` which will use the original (old) backup metadata format. Backups using the old format can always be imported on newer versions of LXD. If the flag is not specified and the server has support for the `backup_metadata_version` API extension, version `2` is used by default.

`--instance-only`
: By default, the export file contains all snapshots of the instance. Add this flag to export the instance without its snapshots.

To create a backup of an instance, send a POST request to the `backups` endpoint:

```
lxc query --request POST /1.0/instances/<instance_name>/backups --data '{"name": "
"}'
```

You can specify a name for the backup, or use the default (`backup0`, `backup1` and so on).

You can add any of the following fields to the request data:

`"compression_algorithm": "bzip2"`
: By default, the output file uses `gzip` compression. You can specify a different compression algorithm (for example, `bzip2`) or turn off compression with `none`.

`"optimized-storage": true`
: If your storage pool uses the `btrfs` or the `zfs` driver, set the `"optimized-storage"` field to `true` to store the data as a driver-specific binary blob instead of an archive of individual files. In this case, the backup can only be used with pools that use the same storage driver.

  Exporting a volume in optimized mode is usually quicker than exporting the individual files. Snapshots are exported as differences from the main volume, which decreases their size (quota) and makes them easily accessible.

`"instance-only": true`
: By default, the backup contains all snapshots of the instance. Set this field to `true` to back up the instance without its snapshots.

After creating the backup, you can download it with the following request:

```
lxc query --request GET /1.0/instances/<instance_name>/backups/<backup_name>/
export > <file_name>
```

Remember to delete the backup when you don't need it anymore:

```
lxc query --request DELETE /1.0/instances/<instance_name>/backups/<backup_name>
```

See `POST /1.0/instances/{name}/backups`, `GET /1.0/instances/{name}/backups/{backup}/export`, and `DELETE /1.0/instances/{name}/backups/{backup}` for more information.

Restore an instance from an export file

You can import an export file (for example, `/path/to/my-backup.tgz`) as a new instance.

CLI

API

To import an export file, use the following command:

```
lxc import <file_path> [<instance_name>]
```

If you do not specify an instance name, the original name of the exported instance is used for the new instance. If an instance with that name already (or still) exists in the specified storage pool, the command returns an error. In that case, either delete the existing instance before importing the backup or specify a different instance name for the import.

Add the `--storage` flag to specify which storage pool to use, or the `--device` flag to override the device configuration (syntax: `--device <device_name>,<device_option>=<value>`).

To import an export file, post it to the `/1.0/instances` endpoint:

```
curl -X POST -H "Content-Type: application/octet-stream" --data-binary @<file_
path> \
--unix-socket /var/snap/lxd/common/lxd/unix.socket lxd/1.0/instances
```

If an instance with that name already (or still) exists in the specified storage pool, the command returns an error. In this case, delete the existing instance before importing the backup.

See `POST /1.0/instances` for more information.

Copy an instance to a backup server

You can copy an instance to a secondary backup server to back it up.

See *Secondary backup LXD server* (page 318) for more information, and *How to migrate LXD instances between servers* (page 135) for instructions.

How to migrate LXD instances between servers

If you use the LXD client, you can migrate or copy instances from one LXD server (remote or local) to another.

> **Note**
>
> *Remote servers* (page 70) are a concept of the LXD client. Therefore, there is no direct equivalent for migrating instances between servers in the API or the UI.
>
> However, you can *export an instance* (page 133) from one server and *import it* (page 134) to another server.

Migrate instances

To migrate an instance (move it from one LXD server to another) using the CLI, use the *`lxc move`* (page 788) command:

```
lxc move [<source_remote>:]<source_instance_name> <target_remote>:[<target_
instance_name>]
```

When migrating a container, you must stop it first. See *Live migration for containers* (page 137) for more information.

When migrating a virtual machine, you must either enable *Live migration for virtual machines* (page 136) or stop it first.

Copy instances

Use the *`lxc copy`* (page 761) command if you want to duplicate the instance instead of migrating it:

```
lxc copy [<source_remote>:]<source_instance_name> <target_remote>:[<target_
instance_name>]
```

If the volume already exists in the target location, use the `--refresh` flag to update the copy. To learn about the benefits, see: *Optimized volume transfer* (page 572).

Migrate and copy options

For both migrating and copying instances, you don't need to specify the source remote if it is your default remote, and you can leave out the target instance name if you want to use the same instance name on the target remote server.

If you want to migrate the instance to a specific cluster member, specify that member's name with the `--target` flag. In this case, do not specify the source and target remote.

You can add the `--mode` flag to choose a transfer mode, depending on your network setup:

`pull` (default)
: Instruct the target server to connect to the source server and pull the respective instance.

`push`
: Instruct the source server to connect to the target server and push the instance.

`relay`
: Instruct the client to connect to both the source and the target server and transfer the data through the client.

If you need to adapt the configuration for the instance to run on the target server, you can either specify the new configuration directly (using `--config`, `--device`, `--storage` or `--target-project`) or through profiles (using `--no-profiles` or `--profile`). See *`lxc move --help`* (page 788) for all available flags.

Live migration

Live migration means migrating an instance to another server while it is running. This method is supported for virtual machines. For containers, there is limited support.

Live migration for virtual machines

Virtual machines can be migrated to another server while they are running, thus avoiding any downtime.

For a virtual machine to be eligible for live migration, it must meet the following criteria:

- It must have support for stateful migration enabled. To enable this, set *`migration.stateful`* (page 429) to `true` on the virtual machine. This setting can only be updated when the machine is stopped. Thus, be sure to configure this setting before you need to live-migrate:

  ```
  lxc config set <instance-name> migration.stateful=true
  ```

  > **Note**
  >
  > When *`migration.stateful`* (page 429) is enabled in LXD, virtiofs shares are disabled, and files are only shared via the 9P protocol. Consequently, guest OSes lacking 9P support, such as CentOS 8, cannot share files with the host unless stateful migration is disabled. Additionally, the `lxd-agent` will not function for these guests under these conditions.

- When using a local pool, the *size.state* (page 484) of the virtual machine's root disk device must be set to at least the size of the virtual machine's *limits.memory* (page 423) setting.

> **Note**
>
> If you are using a remote storage pool like Ceph RBD to back your instance, you don't need to set *size.state* (page 484) to perform live migration.

- The virtual machine must not depend on any resources specific to its current host, such as local storage or a local (non-OVN) bridge network.

Live migration for containers

For containers, there is limited support for live migration using CRIU (Checkpoint/Restore in Userspace)[84]. However, because of extensive kernel dependencies, only very basic containers (non-`systemd` containers without a network device) can be migrated reliably. In most real-world scenarios, you should stop the container, migrate it, then start it again.

If you want to use live migration for containers, you must enable CRIU on both the source and the target server. If you are using the snap, use the following commands to enable CRIU:

```
snap set lxd criu.enable=true
sudo systemctl reload snap.lxd.daemon
```

Otherwise, make sure you have CRIU installed on both systems.

To optimize the memory transfer for a container, set the *migration.incremental.memory* (page 428) property to `true` to make use of the pre-copy features in CRIU. With this configuration, LXD instructs CRIU to perform a series of memory dumps for the container. After each dump, LXD sends the memory dump to the specified remote. In an ideal scenario, each memory dump will decrease the delta to the previous memory dump, thereby increasing the percentage of memory that is already synced. When the percentage of synced memory is equal to or greater than the threshold specified via *migration.incremental.memory.goal* (page 429), or the maximum number of allowed iterations specified via *migration.incremental.memory.iterations* (page 429) is reached, LXD instructs CRIU to perform a final memory dump and transfers it.

Temporarily migrate all instances from a cluster member

For LXD servers that are members of a cluster, you can use the evacuate and restore operations to temporarily migrate all instances from one cluster member to another. These operations can also live-migrate eligible instances.

For more information, see: *Evacuate and restore cluster members* (page 286).

[84] https://criu.org/Main_Page

Related topics

How-to guides:

- *Migrate instances in a cluster* (page 292)
- *Move an instance to another project* (page 167)
- *Import machines to LXD instances* (page 138)
- *Secondary backup LXD server* (page 318)
- *Export an instance* (page 133)
- *Restore an instance from an export file* (page 134)
- *Move or copy storage volumes* (page 205)

How to import instances:

How to import physical or virtual machines to LXD instances

If you have an existing machine, either physical or virtual (VM or container), you can use the `lxd-migrate` tool to create a LXD instance based on your existing disk or image.

The tool copies the provided partition, disk or image to the LXD storage pool of the provided LXD server, sets up an instance using that storage and allows you to configure additional settings for the new instance.

> **Note**
>
> If you want to configure your new instance during the migration process, set up the entities that you want your instance to use before starting the migration process.
>
> By default, the new instance will use the entities specified in the `default` profile. You can specify a different profile (or a profile list) to customize the configuration. See *How to use profiles* (page 97) for more information. You can also override *Instance options* (page 415), the *storage pool* (page 350) to be used and the size for the *storage volume* (page 352), and the *network* (page 210) to be used.
>
> Alternatively, you can update the instance configuration after the migration is complete.

The tool can create both containers and virtual machines:

- When creating a container, you must provide a disk or partition that contains the root file system for the container. For example, this could be the / root disk of the machine or container where you are running the tool.
- When creating a virtual machine, you must provide a bootable disk, partition, or an image in raw, QCOW, QCOW2, VDI, VHDX, or VMDK format. This means that just providing a file system is not sufficient, and you cannot create a virtual machine from a container that you are running. It is also not possible to create a virtual machine from the physical machine that you are using to do the migration, because the migration tool would be using the disk that it is copying. Instead, you could provide a bootable image, or a bootable partition or disk that is currently not in use.

The tool can also inject the required VIRTIO drivers into the image:

- To convert the image into raw format and inject the VIRTIO drivers during the conversion, use the following command:

```
lxd-migrate --conversion=format,virtio
```

Note

The conversion option `virtio` requires `virt-v2v-in-place` to be installed on the host where the LXD server runs.

- For converting Windows images from a foreign hypervisor (not from QEMU/KVM with Q35/`virtio-scsi`), you must install additional drivers on the host:
  - `/usr/share/virtio-win/virtio-win.iso`

    Download `virtio-win.iso`[85].
  - `/usr/share/virt-tools/rhsrvany.exe`
  - `/usr/share/virt-tools/pnp_wait.exe`

    `rhsrvany.exe` and `pnp_wait.exe` are provided in Ubuntu 24.04 and later in the `rhsrvany`[86] package. For other OS versions, download `rhsrvany.exe` and `pnp_wait.exe`[87].

Tip

The `lxd-migrate` command with the `--conversion=format,virtio` option automatically converts the image and injects the VIRTIO drivers during the conversion. However, if you want to manually convert a Windows VM from a foreign hypervisor, you must install both the required Windows drivers (as described above) and `virt-v2v` (>= 2.3.4).

Use `virt-v2v` to convert Windows image into `raw` format and include the required drivers. The resulting image is suitable for use with `lxd-migrate`.

```
# Example 1. Convert a VMDK image to a raw image
sudo virt-v2v --block-driver virtio-scsi -o local -of raw -os ./os -i disk
-if vmdk test-vm-disk.vmdk

# Example 2. Convert a QEMU/KVM qcow2 image to a raw image
sudo virt-v2v --block-driver virtio-scsi -o local -of raw -os ./os -i disk
-if qcow2 test-vm-disk.qcow2

# Example 3. Convert a VMX image to a raw image
sudo virt-v2v --block-driver virtio-scsi -o local -of raw -os ./os -i vmx
./test-vm.vmx
```

You can find the resulting image in the `os` directory and use it with `lxd-migrate` on the next steps. In addition, when migrating already converted images, `lxd-migrate`

[85] https://fedorapeople.org/groups/virt/virtio-win/direct-downloads/stable-virtio/virtio-win.iso
[86] https://launchpad.net/ubuntu/+source/rhsrvany
[87] https://github.com/rwmjones/rhsrvany?tab=readme-ov-file#binary-releases

conversion options are not necessary.

Interactive instance import

Complete the following steps to migrate an existing machine to a LXD instance:

1. Download the `bin.linux.lxd-migrate` tool (`bin.linux.lxd-migrate.aarch64`[88] or `bin.linux.lxd-migrate.x86_64`[89]) from the **Assets** section of the latest LXD release[90].
2. Place the tool on the machine that you want to use to create the instance. Make it executable (usually by running `chmod u+x bin.linux.lxd-migrate`).
3. Make sure that the machine has `rsync` and `file` installed. If they are missing, install them (for example, with `sudo apt install rsync file`).
4. Run the tool:

```
sudo ./bin.linux.lxd-migrate
```

   The tool then asks you to provide the information required for the migration.

   1. Specify the LXD server URL, either as an IP address or as a DNS name.

      Note

      The LXD server must be *exposed to the network* (page 44). If you want to import to a local LXD server, you must still expose it to the network. You can then specify `127.0.0.1` as the IP address to access the local server.

   2. Check and confirm the certificate fingerprint.
   3. Choose a method for authentication (see *Remote API authentication* (page 358)).

      For example, if you choose using a certificate token, log on to the LXD server and create a token for the machine on which you are running the migration tool with *`lxc config trust add`* (page 752). Then use the generated token to authenticate the tool.
   4. Choose whether to create a container or a virtual machine. See *Containers and VMs* (page 346).
   5. Specify a name for the instance that you are creating.
   6. Provide the path to a root file system (for containers) or a bootable disk, partition or image file (for virtual machines).
   7. For containers, optionally add additional file system mounts.
   8. For virtual machines, specify whether secure boot is supported.
   9. Optionally, configure the new instance. You can do so by specifying *profiles* (page 97), directly setting *configuration options* (page 415) or changing *storage* (page 175) or *network* (page 210) settings.

[88] https://github.com/canonical/lxd/releases/latest/download/bin.linux.lxd-migrate.aarch64
[89] https://github.com/canonical/lxd/releases/latest/download/bin.linux.lxd-migrate.x86_64
[90] https://github.com/canonical/lxd/releases

Alternatively, you can configure the new instance after the migration.

10. When you are done with the configuration, start the migration process.

```
~$ sudo ./bin.linux.lxd-migrate Please provide LXD server URL:
https://192.0.2.7:8443Certificate fingerprint: xxxxxxxxxxxxxxxxxok
(y/n)? y 1) Use a certificate token2) Use an existing TLS authentication
certificate3) Generate a temporary TLS authentication certificatePlease
pick an authentication mechanism above: 1Please provide the certificate
token: xxxxxxxxxxxxxxxx Remote LXD server: Hostname: bar Version: 5.4
Would you like to create a container (1) or virtual-machine (2)?: 1Name
of the new instance: fooPlease provide the path to a root filesystem:
/Do you want to add additional filesystem mounts? [default=no]: Instance
to be created: Name: foo Project: default Type: container Source: /
Additional overrides can be applied at this stage:1) Begin the migration
with the above configuration2) Override profile list3) Set additional
configuration options4) Change instance storage pool or volume size5)
Change instance network Please pick one of the options above
[default=1]: 3Please specify config keys and values (key=value ...):
limits.cpu=2 Instance to be created: Name: foo Project: default Type:
container Source: / Config: limits.cpu: "2" Additional overrides can be
applied at this stage:1) Begin the migration with the above
configuration2) Override profile list3) Set additional configuration
options4) Change instance storage pool or volume size5) Change instance
network Please pick one of the options above [default=1]: 4Please
provide the storage pool to use: defaultDo you want to change the
storage volume size? [default=no]: yesPlease specify the storage volume
size: 20GiB Instance to be created: Name: foo Project: default Type:
container Source: / Storage pool: default Storage volume size: 20GiB
Config: limits.cpu: "2" Additional overrides can be applied at this
stage:1) Begin the migration with the above configuration2) Override
profile list3) Set additional configuration options4) Change instance
storage pool or volume size5) Change instance network Please pick one of
the options above [default=1]: 5Please specify the network to use for
the instance: lxdbr0 Instance to be created: Name: foo Project: default
Type: container Source: / Storage pool: default Storage volume size:
20GiB Network name: lxdbr0 Config: limits.cpu: "2" Additional overrides
can be applied at this stage:1) Begin the migration with the above
configuration2) Override profile list3) Set additional configuration
options4) Change instance storage pool or volume size5) Change instance
network Please pick one of the options above [default=1]: 1Instance foo
successfully created
```

```
~$ sudo ./bin.linux.lxd-migrate Please provide LXD server URL:
https://192.0.2.7:8443Certificate fingerprint: xxxxxxxxxxxxxxxxxok
(y/n)? y 1) Use a certificate token2) Use an existing TLS authentication
certificate3) Generate a temporary TLS authentication certificatePlease
pick an authentication mechanism above: 1Please provide the certificate
token: xxxxxxxxxxxxxxxx Remote LXD server: Hostname: bar Version: 5.4
Would you like to create a container (1) or virtual-machine (2)?: 2Name
```

```
of the new instance: fooPlease provide the path to a root filesystem:
./virtual-machine.imgDoes the VM support UEFI Secure Boot? [default=no]:
no Instance to be created: Name: foo Project: default Type:
virtual-machine Source: ./virtual-machine.img Config:
security.secureboot: "false" Additional overrides can be applied at this
stage:1) Begin the migration with the above configuration2) Override
profile list3) Set additional configuration options4) Change instance
storage pool or volume size5) Change instance network Please pick one of
the options above [default=1]: 3Please specify config keys and values
(key=value ...): limits.cpu=2 Instance to be created: Name: foo
Project: default Type: virtual-machine Source: ./virtual-machine.img
Config: limits.cpu: "2" security.secureboot: "false" Additional
overrides can be applied at this stage:1) Begin the migration with the
above configuration2) Override profile list3) Set additional
configuration options4) Change instance storage pool or volume size5)
Change instance network Please pick one of the options above
[default=1]: 4Please provide the storage pool to use: defaultDo you
want to change the storage volume size? [default=no]: yesPlease specify
the storage volume size: 20GiB Instance to be created: Name: foo
Project: default Type: virtual-machine Source: ./virtual-machine.img
Storage pool: default Storage volume size: 20GiB Config: limits.cpu:
"2" security.secureboot: "false" Additional overrides can be applied at
this stage:1) Begin the migration with the above configuration2)
Override profile list3) Set additional configuration options4) Change
instance storage pool or volume size5) Change instance network Please
pick one of the options above [default=1]: 5Please specify the network
to use for the instance: lxdbr0 Instance to be created: Name: foo
Project: default Type: virtual-machine Source: ./virtual-machine.img
Storage pool: default Storage volume size: 20GiB Network name: lxdbr0
Config: limits.cpu: "2" security.secureboot: "false" Additional
overrides can be applied at this stage:1) Begin the migration with the
above configuration2) Override profile list3) Set additional
configuration options4) Change instance storage pool or volume size5)
Change instance network Please pick one of the options above
[default=1]: 1Instance foo successfully created
```

5. When the migration is complete, check the new instance and update its configuration to the new environment. Typically, you must update at least the storage configuration (`/etc/fstab`) and the network configuration.

Non-interactive instance import

Alternatively, the entire instance import configuration can be provided using `lxd-migrate` flags. If any required flag is missing, `lxd-migrate` will interactively prompt for the missing value. However, when the `--non-interactive` flag is used, an error is returned instead.

Note that if any flag contains an invalid value, an error is returned regardless of the mode (interactive or non-interactive).

The `lxd-migrate` command supports the following flags that can be used in non-interactive migration:

```
Instance configuration:
  -c, --config              Config key/value to apply to the new instance
      --mount-path          Additional container mount paths
      --name                Name of the new instance
      --network             Network name
      --no-profiles         Create the instance with no profiles applied
      --profiles            Profiles to apply on the new instance (default
[default])
      --project             Project name
      --source              Path to the root filesystem for containers, or to the
block device or disk image file for virtual machines
      --storage             Storage pool name
      --storage-size        Size of the instance's storage volume
      --type                Type of the instance to create (container or vm)

Target server:
      --server              Unix or HTTPS URL of the target server
      --token               Authentication token for HTTPS remote
      --cert-path           Trusted certificate path
      --key-path            Trusted certificate path

Other:
      --conversion strings  Comma-separated list of conversion options to apply.
Allowed values are: [format, virtio] (default [format])
      --non-interactive     Prevent further interaction if migration questions
are incomplete
      --rsync-args          Extra arguments to pass to rsync
```

Example VM import to local LXD server:

```
lxd-migrate \
  --name v1 \
  --type vm \
  --source "${sourcePath}" \
  --non-interactive
```

Example VM import to remote HTTPS server:

```
# Token from remote server.
token=$(lxc config trust add --name lxd-migrate --quiet)

lxd-migrate \
  --server https://example.com:8443 \
  --token "$token" \
  --name v1 \
  --type vm \
  --source "${sourcePath}" \
  --non-interactive
```

Example VM import with secure boot disabled and custom resource limits:

```
lxd-migrate \
  --name v1 \
  --type vm \
  --source "${sourcePath}" \
  --config security.secureboot=false \
  --config limits.cpu=4 \
  --config limits.memory=4GiB \
  --non-interactive
```

How to pass an NVIDIA GPU to a container with a Docker workload:

How to pass an NVIDIA GPU to a container

Steps

If you have an NVIDIA GPU (either discrete (dGPU) or integrated (iGPU)) and you want to pass the runtime libraries and configuration installed on your host to your container, you should add a *LXD GPU device* (page 491). Consider the following scenario:

Your host is an NVIDIA single board computer that has a Tegra SoC with an iGPU, and you have the Tegra SDK installed on the host. You want to create a LXD container and run an application inside the container using the iGPU as a compute backend. You want to run this application inside a Docker container (or another OCI-compliant runtime). To achieve this, complete the following steps:

1. Running a Docker container inside a LXD container can potentially consume a lot of disk space if the outer container is not well configured. Here are two options you can use to optimize the consumed disk space:

   - Either you create a BTRFS storage pool to back the LXD container so that the Docker image later used does not use the VFS storage driver which is very space inefficient, then you initialize the LXD container with `security.nesting` (page 436) enabled (needed for running a Docker container inside a LXD container) and using the BTRFS storage pool:

     ```
     lxc storage create p1 btrfs size=15GiB
     lxc init ubuntu:24.04 t1 --config security.nesting=true -s p1
     ```

   - Or you use the `overlayFS` storage driver in Docker but you need to specify the following syscall interceptions, still with the `security.nesting` (page 436) enabled:

     ```
     lxc init ubuntu:24.04 t1 --config security.nesting=true --config
     security.syscalls.intercept.mknod=true --config security.syscalls.
     intercept.setxattr=true
     ```

2. Add the GPU device to your container:

   - If you want to do an iGPU pass-through:

     ```
     lxc config device add t1 igpu0 gpu gputype=physical id=nvidia.com/igpu=0
     ```

   - If you want to do a dGPU pass-through:

```
lxc config device add t1 gpu0 gpu gputype=physical id=nvidia.com/gpu=0
```

After adding the device, let's try to run a basic MNIST[91] inference job inside our LXD container.

1. Create a cloud-init script that installs the Docker runtime, the NVIDIA Container Toolkit[92], and a script to run a test TensorRT[93] workload:

```
#cloud-config
package_update: true
write_files:
  # `run_tensorrt.sh` compiles samples TensorRT applications and run the the
`sample_onnx_mnist` program which loads an ONNX model into the TensorRT
inference server and execute a digit recognition job.
  - path: /root/run_tensorrt.sh
    permissions: "0755"
    owner: root:root
    content: |
      #!/bin/bash
      echo "OS release,Kernel version"
      (. /etc/os-release; echo "${PRETTY_NAME}"; uname -r) | paste -s -d,
      echo
      nvidia-smi -q
      echo
      exec bash -o pipefail -c "
      cd /workspace/tensorrt/samples
      make -j4
      cd /workspace/tensorrt/bin
      ./sample_onnx_mnist
      retstatus=\${PIPESTATUS[0]}
      echo \"Test exited with status code: \${retstatus}\" >&2
      exit \${retstatus}
      "
runcmd:
  # Install Docker to run the AI workload
  - curl -fsSL https://get.docker.com -o install-docker.sh
  - sh install-docker.sh --version 24.0
  # The following installs the NVIDIA container toolkit
  # as explained in the official doc website: https://docs.nvidia.com/
datacenter/cloud-native/container-toolkit/latest/install-guide.html
#installing-with-apt
  - curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | gpg --
dearmor -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg
  - curl -fsSL https://nvidia.github.io/libnvidia-container/stable/deb/
nvidia-container-toolkit.list | sed -e 's#deb https://#deb [signed-by=/usr/
share/keyrings/nvidia-container-toolkit-keyring.gpg] https://#g' -e '/
experimental/ s/^#//g' | tee /etc/apt/sources.list.d/nvidia-container-
toolkit.list
```

(continues on next page)

[91] https://en.wikipedia.org/wiki/MNIST_database
[92] https://github.com/NVIDIA/nvidia-container-toolkit
[93] https://github.com/NVIDIA/TensorRT

```
    # Now that an new apt source/key was added, update the package definitions.
    - apt-get update
    # Install NVIDIA container toolkit
    - DEBIAN_FRONTEND=noninteractive apt-get install -y nvidia-container-toolkit
    # Ultimately, we need to tell Docker, our container runtime, to use `nvidia-ctk` as a runtime.
    - nvidia-ctk runtime configure --runtime=docker --config=/etc/docker/daemon.json
    - systemctl restart docker
```

2. Apply this `cloud-init` setup to your instance:

   ```
   lxc config set t1 cloud-init.user-data - < cloud-init.yml
   ```

3. Start the instance:

   ```
   lxc start t1
   ```

4. Wait for the `cloud-init` process to finish:

   ```
   lxc exec t1 -- cloud-init status --wait
   ```

5. Once `cloud-init` is finished, open a shell in the instance:

   ```
   lxc exec t1 -- bash
   ```

6. Edit the NVIDIA container runtime to avoid using `cgroups`:

   ```
   sudo nvidia-ctk config  --in-place --set nvidia-container-cli.no-cgroups
   ```

7. If you use an iGPU and your NVIDIA container runtime is not automatically enabled with CSV mode (needed for NVIDIA Tegra board), enable it manually:

   ```
   sudo nvidia-ctk config --in-place --set nvidia-container-runtime.mode=csv
   ```

8. Now, run the inference workload with Docker:

   - If you set up a dGPU pass-through:

     ```
     docker run --gpus all --runtime nvidia --rm -v $(pwd):/sh_input nvcr.io/nvidia/tensorrt:24.02-py3 bash /sh_input/run_tensorrt.sh
     ```

   - If you set up an iGPU pass-through:

     ```
     docker run --gpus all --runtime nvidia --rm -v $(pwd):/sh_input nvcr.io/nvidia/tensorrt:24.02-py3-igpu bash /sh_input/run_tensorrt.sh
     ```

In the end you should see something like:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@=   ++++#++=*@@@@@
@@@@@@@@#.            *@@@@@
@@@@@@@@=             *@@@@@
@@@@@@@@.   .. ...****%@@@@@
@@@@@@@@: .%@@#@@@@@@@@@@@@@
@@@@@@@%  -@@@@@@@@@@@@@@@@@
@@@@@@@%  -@@*@@@*@@@@@@@@@@
@@@@@@@#  :#- ::. ::=@@@@@@@
@@@@@@@-             -@@@@@@
@@@@@@%.              *@@@@@
@@@@@@#     :==*+==   *@@@@@
@@@@@@%---%%@@@@@@@.  *@@@@@
@@@@@@@@@@@@@@@@@@@+  *@@@@@
@@@@@@@@@@@@@@@@@@@=  *@@@@@
@@@@@@@@@@@@@@@@@@*   *@@@@@
@@@@@%+%@@@@@@@@%.   .%@@@@@
@@@@@*  .******=    -@@@@@@@
@@@@@*             .#@@@@@@@
@@@@@*            =%@@@@@@@@
@@@@@@%#+++=     =@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@

[07/31/2024-13:19:21] [I] Output:
[07/31/2024-13:19:21] [I]  Prob 0  0.0000 Class 0:
[07/31/2024-13:19:21] [I]  Prob 1  0.0000 Class 1:
[07/31/2024-13:19:21] [I]  Prob 2  0.0000 Class 2:
[07/31/2024-13:19:21] [I]  Prob 3  0.0000 Class 3:
[07/31/2024-13:19:21] [I]  Prob 4  0.0000 Class 4:
[07/31/2024-13:19:21] [I]  Prob 5  1.0000 Class 5: **********
[07/31/2024-13:19:21] [I]  Prob 6  0.0000 Class 6:
[07/31/2024-13:19:21] [I]  Prob 7  0.0000 Class 7:
[07/31/2024-13:19:21] [I]  Prob 8  0.0000 Class 8:
[07/31/2024-13:19:21] [I]  Prob 9  0.0000 Class 9:
[07/31/2024-13:19:21] [I]
&&&& PASSED TensorRT.sample_onnx_mnist [TensorRT v8603] # ./sample_onnx_mnist
```

Related topics

- *GPU devices reference* (page 491)
- *Why does my VM stop responding when I try to pass through a GPU?* (page 334)

Related topics

Explanation:

- *Instance types in LXD* (page 347)

Reference:

- *Container runtime environment* (page 398)
- *Instance configuration* (page 414)

2.2.2. Images

The following how-to guides cover common operations related to images.

How to work with existing images:

How to use remote images

The *lxc* (page 690) CLI command is pre-configured with several remote image servers. See *Remote image servers* (page 391) for an overview.

> **Note**
>
> - If you are using the API, you can interact with different LXD servers by using their exposed API addresses. See *Authenticate with the LXD server* (page 45) for instructions on how to authenticate with the servers.
>
>   *How to manage images* (page 150) describes how to interact with images on any LXD server through the API.
>
> - The UI is pre-configured with several remote image servers, but does not currently support adding other servers or managing remote images.
>
>   You can see the available remote images (and which server they are hosted on) when you select the base image for a new instance.

List configured remotes

To see all configured remote servers, enter the following command:

```
lxc remote list
```

Remote servers that use the simple streams format[94] are pure image servers. Servers that use the `lxd` format are LXD servers, which either serve solely as image servers or might provide some images in addition to serving as regular LXD servers. See *Remote server types* (page 391) for more information.

List available images on a remote

To list all remote images on a server, enter the following command:

```
lxc image list <remote>:
```

You can filter the results. See *Filter available images* (page 150) for instructions.

[94] https://git.launchpad.net/simplestreams/tree/

Add a remote server

How to add a remote depends on the protocol that the server uses.

Add a simple streams server

To add a simple streams server as a remote, enter the following command:

```
lxc remote add <remote_name> <URL> --protocol=simplestreams
```

The URL must use HTTPS.

Add a remote LXD server

To add a LXD server as a remote, enter the following command:

```
lxc remote add <remote_name> <IP|FQDN|URL|token> [flags]
```

Some authentication methods require specific flags (for example, use *lxc remote add <remote_name> <IP|FQDN|URL> --auth-type=oidc* (page 872) for OIDC authentication). See *Authenticate with the LXD server* (page 45) and *Remote API authentication* (page 358) for more information.

For example, enter the following command to add a remote through an IP address:

```
lxc remote add my-remote 192.0.2.10
```

You are prompted to confirm the remote server fingerprint and then asked for the token.

Reference an image

To reference an image, specify its remote and its alias or fingerprint, separated with a colon. For example:

```
ubuntu:24.04
ubuntu-minimal:24.04
images:alpine/edge
local:ed7509d7e83f
```

Select a default remote

If you specify an image name without the name of the remote, the default image server is used.

To see which server is configured as the default image server, enter the following command:

```
lxc remote get-default
```

To select a different remote as the default image server, enter the following command:

```
lxc remote switch <remote_name>
```

How to manage images

When working with images, you can inspect various information about the available images, view and edit their properties and configure aliases to refer to specific images. You can also export an image to a file, which can be useful to *copy or import it* (page 156) on another machine.

List available images

CLI

API

UI

To list all images on a server, enter the following command:

```
lxc image list [<remote>:]
```

If you do not specify a remote, the *default remote* (page 149) is used.

Query the `/1.0/images` endpoint to list all images on the server:

```
lxc query --request GET /1.0/images
```

To include information about each image, add `recursion=1`:

```
lxc query --request GET /1.0/images?recursion=1
```

See `GET /1.0/images` and `GET /1.0/images?recursion=1` for more information.

Note

The `/1.0/images` endpoint is available on LXD servers, but not on simple streams servers (see *Remote server types* (page 391)). Public image servers, like the official Ubuntu image server[95], use the simple streams format[96].

To retrieve the list of images from a simple streams server, start at the `streams/v1/index.sjson` index (for example, `https://cloud-images.ubuntu.com/releases/streams/v1/index.sjson`).

Go to *Images* to view all images on the local server.

Filter available images

CLI

API

UI

To filter the results that are displayed, specify a part of the alias or fingerprint after the command. For example, to show all Ubuntu 24.04 LTS images, enter the following command:

[95] https://cloud-images.ubuntu.com/releases/
[96] https://git.launchpad.net/simplestreams/tree/

```
lxc image list ubuntu: 24.04
```

You can specify several filters as well. For example, to show all Arm 64-bit Ubuntu 24.04 LTS images, enter the following command:

```
lxc image list ubuntu: 24.04 arm64
```

To filter for properties other than alias or fingerprint, specify the filter in `<key>=<value>` format. For example:

```
lxc image list ubuntu: 24.04 architecture=x86_64
```

You can *filter* (page 621) the images that are displayed by any of their fields.

For example, to show all Ubuntu images, or all images for Ubuntu 24.04 LTS:

```
lxc query --request GET /1.0/images?filter=properties.os+eq+ubuntu
lxc query --request GET /1.0/images?filter=properties.version+eq+24.04
```

You can specify several filters as well. For example, to show all Arm 64-bit images for virtual machines, enter the following command:

```
lxc query --request GET /1.0/images?
filter=architecture+eq+arm64+and+type+eq+virtual-machine
```

You can also use a regular expression:

```
lxc query --request GET "/1.0/images?filter=fingerprint+eq+be25.*"
```

See `GET /1.0/images` and *Filtering* (page 621) for more information.

To filter the images that are displayed, use the search box.

For example, to show all Ubuntu images, search for `ubuntu`. To display only images for version 24.04, search for `24.04`.

View image information

CLI

API

UI

To view information about an image, enter the following command:

```
lxc image info <image_ID>
```

As the image ID, you can specify either the image's alias or its fingerprint. For a remote image, remember to include the remote server (for example, `ubuntu:24.04`).

To display only the image properties, enter the following command:

```
lxc image show <image_ID>
```

You can also display a specific image property (located under the `properties` key) with the following command:

```
lxc image get-property <image_ID> <key>
```

For example, to show the release name of the official Ubuntu 24.04 LTS image, enter the following command:

```
lxc image get-property ubuntu:24.04 release
```

To view all information about an image, query it using its fingerprint:

```
lxc query --request GET /1.0/images/<fingerprint>
```

See `GET /1.0/images/{fingerprint}` for more information.

If you don't know the fingerprint but the alias, you can retrieve the fingerprint by querying the `/1.0/images/aliases/{alias}` endpoint:

```
lxc query --request GET /1.0/images/aliases/<alias>
```

See `GET /1.0/images/aliases/{name}` for more information.

The UI does not currently support viewing detailed image information.

Edit image properties

CLI

API

UI

To set a specific image property that is located under the `properties` key, enter the following command:

```
lxc image set-property <image_ID> <key> <value>
```

> **Note**
>
> These properties can be used to convey information about the image. They do not configure LXD's behavior in any way.

To edit the full image properties, including the top-level properties, enter the following command:

```
lxc image edit <image_ID>
```

To set a specific image property that is located under the `properties` key, send a PATCH request to the image:

```
lxc query --request PATCH /1.0/images/<fingerprint> --data '{
  "properties": {
    "<key>": "<value>"
  }
}'
```

See `PATCH /1.0/images/{fingerprint}` for more information.

> **Note**
>
> These properties can be used to convey information about the image. They do not configure LXD's behavior in any way.

To update the full image properties, including the top-level properties, send a PUT request with the full image data:

```
lxc query --request PUT /1.0/images/<fingerprint> --data '<image_configuration>'
```

See `PUT /1.0/images/{fingerprint}` for more information.

The UI does not currently support editing image properties.

Delete an image

CLI

API

UI

To delete a local copy of an image, enter the following command:

```
lxc image delete <image_ID>
```

To delete a local copy of an image, send a DELETE request:

```
lxc query --request DELETE /1.0/images/<fingerprint>
```

See `DELETE /1.0/images/{fingerprint}` for more information.

In the images list, click the *Delete* button () next to an image to delete it.

You can also select several images and click the *Delete images* button at the top to delete all selected images.

Deleting an image won't affect running instances that are already using it, but it will remove the image locally.

After deletion, if the image was downloaded from a remote server, it will be removed from local cache and downloaded again on next use. However, if the image was manually created (not cached), the image will be deleted.

Configure image aliases

Configuring an alias for an image can be useful to make it easier to refer to an image, since remembering an alias is usually easier than remembering a fingerprint. Most importantly, however, you can change an alias to point to a different image, which allows creating an alias that always provides a current image (for example, the latest version of a release).

CLI

API

UI

You can see some of the existing aliases in the image list. To see the full list, enter the following command:

```
lxc image alias list
```

You can directly assign an alias to an image when you *copy or import* (page 156) or *publish* (page 159) it. Alternatively, enter the following command:

```
lxc image alias create <alias_name> <image_fingerprint>
```

You can also delete an alias:

```
lxc image alias delete <alias_name>
```

To rename an alias, enter the following command:

```
lxc image alias rename <alias_name> <new_alias_name>
```

If you want to keep the alias name, but point the alias to a different image (for example, a newer version), you must delete the existing alias and then create a new one.

To retrieve a list of all defined aliases, query the `/1.0/images/aliases` endpoint:

```
lxc query --request GET /1.0/images/aliases
```

To include information about each alias, add `recursion=1`:

```
lxc query --request GET /1.0/images/aliases?recursion=1
```

See `GET /1.0/images/aliases` and `GET /1.0/images/aliases?recursion=1` for more information.

You can directly assign an alias to an image when you *copy or import* (page 156) or *publish* (page 159) it. Alternatively, send a POST request to the `/1.0/images/aliases` endpoint to create an alias:

```
lxc query --request POST /1.0/images/aliases --data '{
  "name": "<alias_name>",
  "target": "<image_fingerprint>"
}'
```

See `POST /1.0/images/aliases` for more information.

You can also delete an alias:

```
lxc query --request DELETE /1.0/images/aliases/<alias_name>
```

To rename an alias, send a POST request to the alias:

```
lxc query --request POST /1.0/images/aliases/<alias_name> --data '{
  "name": "<new_alias_name>"
}'
```

If you want to keep the alias name, but point the alias to a different image (for example, a newer version), send a PATCH request to the alias:

```
lxc query --request PATCH /1.0/images/aliases/<alias_name> --data '{
  "target": "<new_fingerprint>"
}'
```

See `DELETE /1.0/images/aliases/{name}`, `POST /1.0/images/aliases/{name}`, and `PATCH /1.0/images/aliases/{name}` for more information.

The UI displays configured aliases in the images list, but it does not currently support configuring image aliases.

Export an image to a set of files

Images are located in the image store of your local server or a remote LXD server. You can export them to a file or a set of files though (see *Image tarballs* (page 394)). This method can be useful to back up image files or to transfer them to an air-gapped environment.

CLI

API

UI

To export a container image to a set of files, enter the following command:

```
lxc image export [<remote>:]<image> [<output_directory_path>]
```

To export a virtual machine image to a set of files, add the `--vm` flag:

```
lxc image export [<remote>:]<image> [<output_directory_path>] --vm
```

Send a query to the `export` endpoint of the image to retrieve it:

```
curl -X GET --unix-socket /var/snap/lxd/common/lxd/unix.socket lxd/1.0/images/
<fingerprint>/export \
--output <output-file>
```

If the image is a *split image* (page 395), the output file contains two separate tarballs in multipart format.

See `GET /1.0/images/{fingerprint}/export` for more information.

The UI does not currently support exporting images.

See *Image format* (page 392) for a description of the file structure used for the image.

How to associate profiles with an image

You can associate one or more profiles with a specific image. Instances that are created from the image will then automatically use the associated profiles in the order they were specified.

To associate a list of profiles with an image, add the profiles to the image configuration in the `profiles` section (see *Edit image properties* (page 152)).

CLI

API

UI

Use the *lxc image edit* (page 775) command to edit the profiles section:

```yaml
profiles:
- default
```

To update the full image properties, including the profiles section, send a PUT request with the full image data:

```
lxc query --request PUT /1.0/images/<fingerprint> --data '<image_configuration>'
```

See `PUT /1.0/images/{fingerprint}` for more information.

The UI does not currently support editing the image configuration. Therefore, you cannot associate profiles with an image through the UI.

Most provided images come with a profile list that includes only the `default` profile. To prevent any profile (including the `default` profile) from being associated with an image, pass an empty list.

> **Note**
>
> Passing an empty list is different than passing `nil`. If you pass `nil` as the profile list, only the `default` profile is associated with the image.

You can override the associated profiles for an image when creating an instance by adding the `--profile` or the `--no-profiles` flag to the launch or init command (when using the CLI), or by specifying a list of profiles in the request data (when using the API).

How to import and create images:

How to copy and import images

To add images to an image store, you can either copy them from another server or import them from files (either local files or files on a web server).

> **Note**
>
> The UI does not currently support copying or importing images.
>
> There is support for importing custom ISO files, but these ISO files are different from images. When you create an instance from a custom ISO file, the ISO file is mounted as a storage volume in a new empty VM, and you can then install the VM from the ISO file. See *Content type iso* (page 352) and *Create a VM that boots from an ISO* (page 80) for more information.

Copy an image from a remote

CLI

API

To copy an image from one server to another, enter the following command:

```
lxc image copy [<source_remote>:]<image> <target_remote>:
```

Note

To copy the image to your local image store, specify `local:` as the target remote.

See *lxc image copy --help* (page 774) for a list of all available flags. The most relevant ones are:

--alias
: Assign an alias to the copy of the image.

--copy-aliases
: Copy the aliases that the source image has.

--auto-update
: Keep the copy up-to-date with the original image.

--vm
: When copying from an alias, copy the image that can be used to create virtual machines.

To copy an image from one server to another, *export it to your local machine* (page 155) and then *import it to the other server* (page 157).

Import an image from files

If you have image files that use the required *Image format* (page 392), you can import them into your image store.

There are several ways of obtaining such image files:

- Exporting an existing image (see *Export an image to a set of files* (page 155))
- Building your own image using LXD image builder (see *Build an image* (page 161))
- Downloading image files from a *remote image server* (page 391) (note that it is usually easier to *use the remote image* (page 148) directly instead of downloading it to a file and importing it)

Import from the local file system

CLI

API

To import an image from the local file system, use the *lxc image import* (page 777) command. This command supports both *unified images* (page 394) (compressed file or directory) and *split images* (page 395) (two files).

To import a unified image from one file or directory, enter the following command:

```
lxc image import <image_file_or_directory_path> [<target_remote>:]
```

To import a split image, enter the following command:

```
lxc image import <metadata_tarball_path> <rootfs_tarball_path> [<target_remote>:]
```

In both cases, you can assign an alias with the `--alias` flag. See *lxc image import --help* (page 777) for all available flags.

To import an image from the local file system, send a POST request to the `/1.0/images` endpoint.

For example, to import a unified image from one file:

```
curl -X POST -H 'Content-Type: application/octet-stream' --unix-socket /var/snap/
lxd/common/lxd/unix.socket lxd/1.0/images \
--data-binary @<image_file_path>
```

To import a split image from a metadata file and a container `rootfs` file:

```
curl -X POST -H 'Content-Type: multipart/form-data' --unix-socket /var/snap/lxd/
common/lxd/unix.socket lxd/1.0/images \
--form metadata=@<metadata_tarball_path> --form rootfs=@<rootfs_tarball_path>
```

To import a split image from a metadata file and a VM `rootfs.img` file:

```
curl -X POST -H 'Content-Type: multipart/form-data' --unix-socket /var/snap/lxd/
common/lxd/unix.socket lxd/1.0/images \
--form metadata=@<metadata_tarball_path> --form rootfs.img=@<rootfs_tarball_path>
```

> **Note**
>
> For a split image, you must send the metadata tarball first and the rootfs image after.

See `POST /1.0/images` for more information.

Import from a file on a remote web server

You can import image files from a remote web server by URL. This method is an alternative to running a LXD server for the sole purpose of distributing an image to users. It only requires a basic web server with support for custom headers (see *Custom HTTP headers* (page 159)).

The image files must be provided as unified images (see *Unified tarball* (page 394)).

CLI

API

To import an image file from a remote web server, enter the following command:

```
lxc image import <URL>
```

You can assign an alias to the local image with the `--alias` flag.

To import an image file from a remote web server, send a POST request with the image URL to the /1.0/images endpoint:

```
lxc query --request POST /1.0/images --data '{
  "source": {
    "type": "url",
    "url": "<URL>"
  }
}'
```

See POST /1.0/images for more information.

Custom HTTP headers

LXD requires the following custom HTTP headers to be set by the web server:

LXD-Image-Hash
: The SHA256 of the image that is being downloaded.

LXD-Image-URL
: The URL from which to download the image.

LXD sets the following headers when querying the server:

LXD-Server-Architectures
: A comma-separated list of architectures that the client supports.

LXD-Server-Version
: The version of LXD in use.

How to create images

If you want to create and share your own images, you can do this either based on an existing instance or snapshot or by building your own image from scratch.

Publish an image from an instance or snapshot

If you want to be able to use an instance or an instance snapshot as the base for new instances, you should create and publish an image from it.

When publishing an image from an instance, make sure that the instance is stopped.

CLI

API

UI

To publish an image from an instance, enter the following command:

```
lxc publish <instance_name> [<remote>:]
```

To publish an image from a snapshot, enter the following command:

```
lxc publish <instance_name>/<snapshot_name> [<remote>:]
```

In both cases, you can specify an alias for the new image with the `--alias` flag, set an expiration date with `--expire` and make the image publicly available with `--public`. If an image with the same name already exists, add the `--reuse` flag to overwrite it. See *lxc publish --help* (page 869) for a full list of available flags.

To publish an image from an instance or a snapshot, send a POST request with the suitable source type to the `/1.0/images` endpoint.

To publish an image from an instance:

```
lxc query --request POST /1.0/images --data '{
  "source": {
    "name": "<instance_name>",
    "type": "instance"
  }
}'
```

To publish an image from a snapshot:

```
lxc query --request POST /1.0/images --data '{
  "source": {
    "name": "<instance_name>/<snapshot_name>",
    "type": "snapshot"
  }
}'
```

In both cases, you can include additional configuration (for example, you can include aliases, set a custom expiration date, or make the image publicly available). For example:

```
lxc query --request POST /1.0/images --data '{
  "aliases": [ { "name": "<alias>" } ],
  "expires_at": "2025-03-23T20:00:00-04:00",
  "public": true,
  "source": {
    "name": "<instance_name>",
    "type": "instance"
  }
}'
```

See `POST /1.0/images` for more information.

The UI does not currently support publishing an image from an instance, but you can publish from a snapshot.

To do so, go to the instance detail page and switch to the *Snapshots* tab. Then click the *Create image* button () and optionally enter an alias for the new image. You can also choose whether the image should be publicly available.

Publishing the image might take a few minutes. You can check the status under *Operations*.

The publishing process can take quite a while because it generates a tarball from the instance or snapshot and then compresses it. As this can be particularly I/O and CPU intensive, publish operations are serialized by LXD.

Prepare the instance for publishing

Before you publish an image from an instance, clean up all data that should not be included in the image. Usually, this includes the following data:

- Instance metadata (use `lxc config metadata` (page 745) or `PATCH /1.0/instances/{name}/metadata`/`PUT /1.0/instances/{name}/metadata` to edit)
- File templates (use `lxc config template` (page 748) or `POST /1.0/instances/{name}/metadata/templates` to edit)
- Instance-specific data inside the instance itself (for example, host SSH keys and `dbus/systemd machine-id`)

Build an image

For building your own images, you can use LXD image builder[97].

See the LXD image builder documentation[98] for instructions for installing and using the tool.

Repack a Windows image

You can run Windows VMs in LXD. To do so, you must repack the Windows ISO with LXD image builder.

See the LXD image builder tutorial[99] for instructions, or How to install a Windows 11 VM using LXD[100] for a full walk-through.

Related topics

Explanation:

- *Local and remote images* (page 348)

Reference:

- *Image format* (page 392)
- *Remote image servers* (page 391)

2.2.3. Projects

The following how-to guides cover common operations related to projects:

How to create and configure projects

You can configure projects at creation time or later. However, note that it is not possible to modify the features that are enabled for a project when the project contains instances.

Create a project

CLI

API

[97] https://github.com/canonical/lxd-imagebuilder
[98] https://canonical-lxd-imagebuilder.readthedocs-hosted.com/en/latest/
[99] https://canonical-lxd-imagebuilder.readthedocs-hosted.com/en/latest/tutorials/use/
[100] https://ubuntu.com/tutorials/how-to-install-a-windows-11-vm-using-lxd

UI

To create a project, use the *lxc project create* (page 862) command.

You can specify configuration options by using the `--config` flag. See *Project configuration* (page 509) for the available configuration options.

For example, to create a project called `my-project` that isolates instances, but allows access to the default project's images and profiles, enter the following command:

```
lxc project create my-project --config features.images=false --config features.
profiles=false
```

To create a project called `my-restricted-project` that blocks access to security-sensitive features (for example, container nesting) but allows snapshots, enter the following command:

```
lxc project create my-restricted-project --config restricted=true --config
restricted.snapshots=allow
```

To create a project, send a POST request to the `/1.0/projects` endpoint.

You can specify configuration options under the `"config"` field. See *Project configuration* (page 509) for the available configuration options.

For example, to create a project called `my-project` that isolates instances, but allows access to the default project's images and profiles, send the following request:

```
lxc query --request POST /1.0/projects --data '{
  "config": {
    "features.images": "false",
    "features.profiles": "false"
  },
  "name": "my-project"
}'
```

To create a project called `my-restricted-project` that blocks access to security-sensitive features (for example, container nesting) but allows snapshots, send the following request:

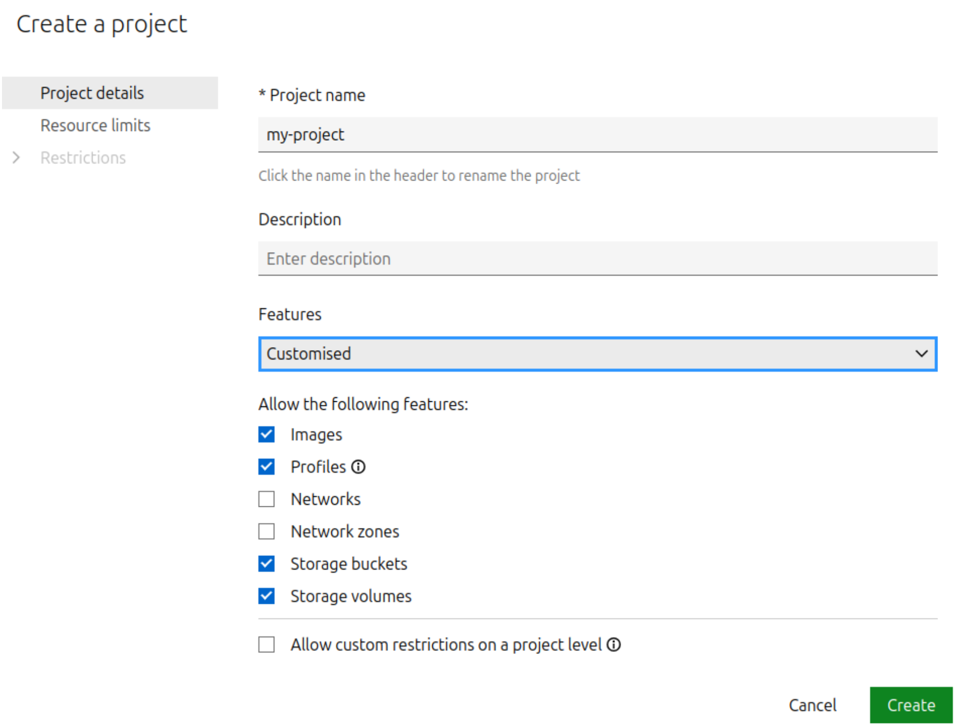
```
lxc query --request POST /1.0/projects --data '{
  "config": {
    "restricted": "true",
    "restricted.snapshots": "allow"
  },
  "name": "my-restricted-project"
}'
```

See `POST /1.0/projects` for more information.

To create a project, expand the *Project* drop-down and select *+ Create project* at the bottom.

Enter a name and optionally a description for the new project. You can create the project using the default set of features or select *Customised* to add or remove specific features. See *Project features* (page 509) for more information about the available features.

For example, to create a project called `my-project` that isolates instances, but allows access to the default project's images and profiles:

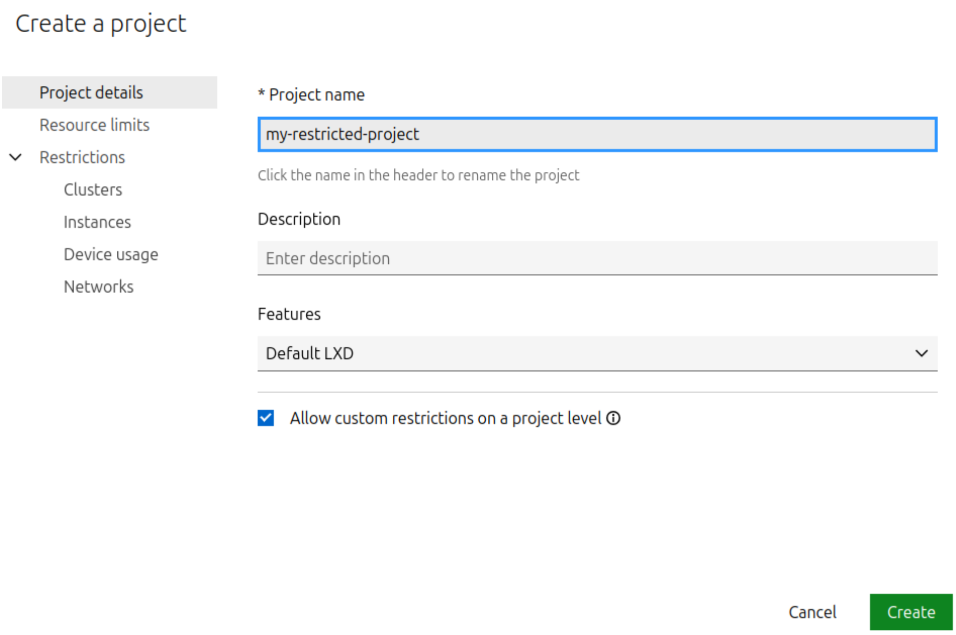


To configure resource limits for the project, select *Resource limits*.

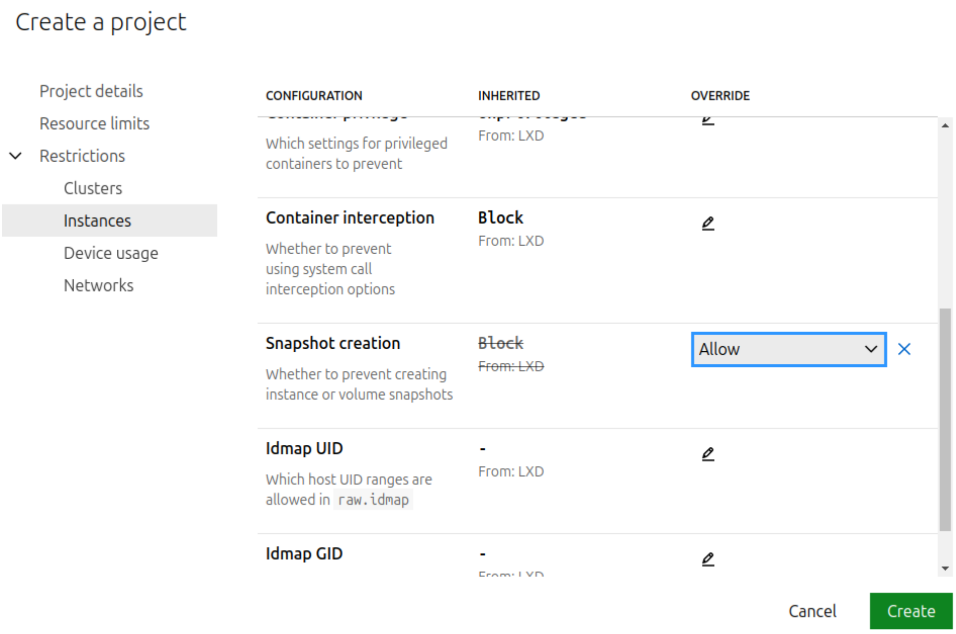
To restrict a project from accessing security-sensitive features, check *Allow custom restrictions on a project level*. You can then configure the restrictions under *Restrictions*. See *Project restrictions* (page 513) for more information.

For example, to create a project called `my-restricted-project` that blocks access to security-sensitive features (for example, container nesting) but allows snapshots:

1. Check *Allow custom restrictions on a project level*:



2. Configure the restrictions:



> **Tip**
>
> When you create a project with the default options, *features.profiles* (page 510) is set to `true`, which means that profiles are isolated in the project.
>
> Consequently, the new project does not have access to the `default` profile of the `default` project and therefore misses required configuration for creating instances (like the root disk). To fix this, add a root disk device to the project's `default` profile (see *Set specific options for a profile* (page 98) for instructions).

Configure a project

To configure a project, you can either set a specific configuration option or edit the full project.

Some configuration options can only be set for projects that do not contain any instances.

Set specific configuration options

CLI

API

UI

To set a specific configuration option, use the *lxc project set* (page 866) command.

For example, to limit the number of containers that can be created in `my-project` to five, enter the following command:

```
lxc project set my-project limits.containers=5
```

To unset a specific configuration option, use the *lxc project unset* (page 868) command.

Note

If you unset a configuration option, it is set to its default value. This default value might differ from the initial value that is set when the project is created.

To set a specific configuration option, send a PATCH request to the project.

For example, to limit the number of containers that can be created in `my-project` to five, send the following request:

```
lxc query --request PATCH /1.0/projects/my-project --data '{
  "config": {
    "limits.containers": "5"
  }
}'
```

See `PATCH /1.0/projects/{name}` for more information.

To update the project configuration, select the respective project from the *Project* drop-down. Then go to *Configuration* and click *Edit configuration* to set or unset any configuration options.

Edit the project

CLI

API

UI

To edit the full project configuration, use the *lxc project edit* (page 863) command. For example:

```
lxc project edit my-project
```

To update the entire project configuration, send a PUT request to the project. For example:

```
lxc query --request PUT /1.0/projects/my-project --data '{
  "config": { ... },
  "description": "<description>"
}'
```

See `PUT /1.0/projects/{name}` for more information.

The UI does not currently support editing the full YAML configuration for a project. However, you can update several or all configuration options at the same time through the UI.

How to work with different projects

If you have more projects than just the `default` project, you must make sure to use or address the correct project when working with LXD.

> **Note**
>
> If you have projects that are *confined to specific users* (page 369), only users with full access to LXD can see all projects.
>
> Users without full access can only see information for the projects to which they have access.

List projects

CLI

API

UI

To list all projects (that you have permission to see), enter the following command:

```
lxc project list
```

By default, the output is presented as a list:

```
~$ lxc project list
+---------------------+-------+---------+---------------+----------------+
NAME | IMAGES | PROFILES | STOR-
AGE VOLUMES | STORAGE BUCKETS | NETWORKS | NETWORK ZONES | DESCRIPTION | USED BY
|+--------------------+-------+---------+---------------+----------------
default | YES | YES | YES | YES | YES | YES | Default LXD project | 19
|+--------------------+-------+---------+---------------+----------------
my-project (current) | YES | NO | NO | NO | YES | YES | | 0
|+--------------------+-------+---------+---------------+----------------
```

You can request a different output format by adding the `--format` flag. See *`lxc project list --help`* (page 865) for more information.

To list all projects (that you have permission to see), send the following request:

```
lxc query --request GET /1.0/projects
```

To display information about each project, use *Recursion* (page 621):

```
lxc query --request GET /1.0/projects?recursion=1
```

See `GET /1.0/projects` and `GET /1.0/projects?recursion=1` for more information.

To list all projects (that you have permission to see), expand the *Project* drop-down.

Switch projects

CLI

API

UI

By default, all commands that you issue in LXD affect the project that you are currently using. To see which project you are in, use the *lxc project list* (page 865) command.

To switch to a different project, enter the following command:

```
lxc project switch <project_name>
```

The API does not have the concept of switching projects. All requests target the default project unless a different project is specified (see *Target a project* (page 167)).

To switch to another project, select a different project from the *Project* drop-down.

Target a project

When using the CLI or the API, you can target a specific project when running a command. Many LXD commands support the `--project` flag or the `project` parameter to run an action in a different project.

> **Note**
>
> You can target only projects that you have permission for.

An example for targeting another project instead of switching to it is listing the instances in a specific project:

CLI

API

UI

To list the instances in a specific project, add the `--project` flag to the *lxc list* (page 785) command. For example:

```
lxc list --project my-project
```

To list the instances in a specific project, add the `project` parameter to the request. For example:

```
lxc query --request GET /1.0/instances?project=my-project
```

Or with *Recursion* (page 621):

```
lxc query --request GET /1.0/instances?recursion=2\&project=my-project
```

The UI does not currently support targeting another project. Instead, *switch to the other project* (page 166).

Move an instance to another project

CLI

API

UI

To move an instance from one project to another, enter the following command:

```
lxc move <instance_name> <new_instance_name> --project <source_project> --target-project <target_project>
```

You can keep the same instance name if no instance with that name exists in the target project.

For example, to move the instance `my-instance` from the `default` project to `my-project` and keep the instance name, enter the following command:

```
lxc move my-instance my-instance --project default --target-project my-project
```

To move an instance from one project to another, send a POST request to the instance:

```
lxc query --request POST /1.0/instances/<instance_name>?project=<source_project> --data '{
  "name": "<new_instance_name>",
  "project": "<target_project>",
  "migration": true
}'
```

If no instance with that name exists in the target project, you can leave out the name for the new instance to keep the existing name.

For example, to move the instance `my-instance` from the `default` project to `my-project` and keep the instance name, enter the following command:

```
lxc query --request POST /1.0/instances/my-instance?project=default --data '{
  "project": "my-project",
  "migration": true
}'
```

Depending on your projects, you might need to change other configuration options when moving the instance. For example, you might need to change the root disk device if one of the projects uses isolated storage volumes.

See `POST /1.0/instances/{name}` for more information.

The UI does not currently support moving instances between projects.

Copy a profile to another project

If you create a project with the default settings, profiles are isolated in the project (*`features.profiles`* (page 510) is set to `true`). Therefore, the project does not have access to the default profile (which is part of the `default` project), and you will see an error similar to the following when trying to create an instance:

```
Error: Failed instance creation: Failed creating instance record: Failed
initialising instance: Failed getting root disk: No root device could be found
```

To fix this, you can copy the contents of the `default` project's default profile into the current project's default profile. To do so:

CLI

API

UI

Enter the following command:

```
lxc profile show default --project default | lxc profile edit default
```

Send the following request, replacing `<project>` with the new project that has an empty default profile:

```
lxc query --request PUT /1.0/profiles/default?projects=<project> --data \
  "$(lxc query --request GET /1.0/profiles/default)"
```

1. Select the `default` project from the *Project* drop-down.
2. Go to *Profiles* and select the default profile.
3. In the profile view, switch to the *Configuration* tab.
4. Select *YAML configuration* and copy the YAML representation of the profile.
5. Select the project with the empty default profile from the *Project* drop-down.
6. Go to *Profiles* and select the empty default profile for the project.
7. In the profile view, switch to the *Configuration* tab.
8. Select *YAML configuration* and click *Edit profile*.
9. Paste the YAML representation that you copied and save the changes.

How to confine users to specific projects

You restrict users or clients to specific projects. Projects can be configured with features, limits, and restrictions to prevent misuse. See *Instances grouping with projects* (page 368) for more information.

How to confine users to specific projects depends on whether LXD is accessible via the *HTTPS API* (page 169), or via the *Unix socket* (page 175).

Confine users to specific projects on the HTTPS API

You can confine access to specific projects by restricting the TLS client certificate that is used to connect to the LXD server. See *Restricted TLS certificates* (page 364) for more information. Only certificates returned by `lxc config trust list` can be managed in this way.

> **Note**
>
> The UI does not currently support configuring project confinement for certificates of this type. Use the CLI or API to set up confinement.

You can also confine access to specific projects via group membership and *Fine-grained authorization* (page 364). The permissions of OIDC clients and fine-grained TLS identities must be managed with `lxc auth` subcommands and the `/1.0/auth` API.

To create a TLS client and restrict the client to a single project, follow these instructions:

CLI

API

Create a restricted trust store entry with access to a project

If you're using token authentication:

```
lxc config trust add --projects <project_name> --restricted
```

To add the client certificate directly:

```
lxc config trust add <certificate_file> --projects <project_name> --restricted
```

The client can then add the server as a remote in the usual way (*lxc remote add <server_name> <token>* (page 872) or *lxc remote add <server_name> <server_address>* (page 872)) and can only access the project or projects that have been specified.

> **Note**
>
> You can specify the `--project` flag when adding a remote. This configuration pre-selects the specified project. However, it does not confine the client to this project.

Create a fine-grained TLS identity with access to a project

First create a group and grant the group the `operator` entitlement on the project.

```
lxc auth group create <group_name>
lxc auth group permission add <group_name> project <project_name> operator
```

The `operator` entitlement grants members of the group permission to create and edit resources belonging to that project, but does not grant permission to delete the project or edit its configuration. See *Fine-grained authorization* (page 364) for more details.

Next create a TLS identity and add the identity to the group:

```
lxc auth identity create tls/<client_name> [<certificate_file>] --group <group_
name>
```

If `<certificate_file>` is provided the identity will be created directly. Otherwise, a token will be returned that the client can use to add the LXD server as a remote:

```
# Client machine
lxc remote add <remote_name> <token>
```

The client will be prompted with a list of projects to use as their default project. Only the configured project will be presented to the client.

Create a restricted trust store entry with access to a project

If you're using token authentication, create the token first:

```
lxc query --request POST /1.0/certificates --data '{
  "name": "<client_name>",
  "projects": ["<project_name>"]
  "restricted": true,
  "token": true,
  "type": "client"
}'
```

See `POST /1.0/certificates` for more information.

The return value of this query contains an operation that has the information that is required to generate the trust token:

```
{
 "class": "token",
 ...
 "metadata": {
    "addresses": [
       "<server_address>"
    ],
    "fingerprint": "<fingerprint>",
    ...
    "secret": "<secret>"
 },
 ...
}
```

Use this information to generate the trust token:

```
echo -n '{"client_name":"<client_name>","fingerprint":"<fingerprint>",'\
'"addresses":["<server_address>"],'\
'"secret":"<secret>","expires_at":"0001-01-01T00:00:00Z"}' | base64 -w0
```

To instead add the client certificate directly, send the following request:

```
lxc query --request POST /1.0/certificates --data '{
  "certificate": "<certificate>",
  "name": "<client_name>",
  "projects": ["<project_name>"]
  "restricted": true,
  "token": false,
  "type": "client"
}'
```

The client can then authenticate using this trust token or client certificate and can only access the project or projects that have been specified.

On the client, generate a certificate to use for the connection:

```
openssl req -x509 -newkey rsa:2048 -keyout "<keyfile_name>" -nodes \
-out "<crtfile_name>" -subj "/CN=<client_name>"
```

Then send a POST request to the `/1.0/certificates?public` endpoint to authenticate:

```
curl -k -s --key "<keyfile_name>" --cert "<crtfile_name>" \
-X POST https://<server_address>/1.0/certificates \
--data '{ "trust_token": "<trust_token>" }'
```

See `POST /1.0/certificates?public` for more information.

Create a fine-grained TLS identity with access to a project

First create a group and grant the group the `operator` entitlement on the project.

```
lxc query --request POST /1.0/auth/groups --data '{
  "name": "<group_name>",
}'

lxc query --request PUT /1.0/auth/groups/<group_name> --data '{
  "permissions": [
    {
      "entity_type": "project",
      "url": "/1.0/projects/<project_name>",
      "entitlement": "operator"
    }
  ]
}'
```

The `operator` entitlement grants members of the group permission to create and edit resources belonging to that project, but does not grant permission to delete the project or edit its configuration. See *Fine-grained authorization* (page 364) for more details.

Next create a TLS identity and add the identity to the group:

```
lxc query --request POST /1.0/auth/identities/tls --data '{
  "name": "<client_name>",
  "groups": ["<group_name>"],
  "token": true
}'
```

See `POST /1.0/auth/identities/tls` for more information.

The return value of this query contains the information that is required to generate the trust token:

```
{
    "client_name": "<client_name>",
    "addresses": [
        "<server_address>"
    ],
    "expires_at": "<expiry_date>"
    "fingerprint": "<fingerprint>",
    "type": "<type>",
    "secret": "<secret>"
}
```

Use this information to generate the trust token:

```
echo -n '{"client_name":"<client_name>","fingerprint":"<fingerprint>",'\
'"addresses":["<server_address>"],'\
'"secret":"<secret>","expires_at":"0001-01-01T00:00:00Z","type":"<type>"}' |
base64 -w0
```

To instead add the client certificate directly, send the following request:

```
lxc query --request POST /1.0/certificates --data '{
  "certificate": "<base64 encoded x509 certificate>",
  "name": "<client_name>",
  "groups": ["<group_name>"]
}'
```

If the certificate was added directly, the client is now authenticated with LXD. If a token was used, the client must use it to add their certificate.

On the client, generate a certificate to use for the connection:

```
openssl req -x509 -newkey rsa:2048 -keyout "<keyfile_name>" -nodes \
-out "<crtfile_name>" -subj "/CN=<client_name>"
```

Send a POST request to the `/1.0/auth/identities/tls?public` endpoint to authenticate:

```
curl --insecure --key "<keyfile_name>" --cert "<crtfile_name>" \
-X POST https://<server_address>/1.0/auth/identities/tls \
--data '{ "trust_token": "<trust_token>" }'
```

See `POST /1.0/auth/identities/tls?public` for more information.

To confine access for an existing certificate:

CLI

API

Trust store entry

Use the following command:

```
lxc config trust edit <fingerprint>
```

Make sure that `restricted` is set to `true` and specify the projects that the certificate should give access to under `projects`.

Fine-grained TLS or OIDC identity

Create a group with the `operator` entitlement on the project:

```
lxc auth group create <group_name>
lxc auth group permission add <group_name> project <project_name> operator
```

Then add the group to the identity. For TLS identities run:

```
lxc auth identity group add tls/<client_name> <group_name>
```

The `<client_name>` must be unique. If it is not, the certificate fingerprint of the client can be used.

For OIDC identities, run:

```
lxc auth identity group add oidc/<client_name> <group_name>
```

The `<client_name>` must be unique. If it is not, the email address of the client can be used.

Trust store entry

Send the following request:

```
lxc query --request PATCH /1.0/certificates/<fingerprint> --data '{
  "projects": ["<project_name>"],
  "restricted": true
}'
```

Make sure that `restricted` is set to `true` and specify the projects that the certificate should give access to under `projects`.

Fine-grained TLS or OIDC identity

Create a group with the `operator` entitlement on the project:

```
lxc query --request POST /1.0/auth/groups --data '{
  "name": "<group_name>",
}'

lxc query --request PUT /1.0/auth/groups/<group_name> --data '{
  "permissions": [
    {
      "entity_type": "project",
      "url": "/1.0/projects/<project_name>",
      "entitlement": "operator"
    }
  ]
}'
```

Then add the group to the identity. For TLS identities run:

```
lxc query --request PATCH /1.0/auth/identities/tls/<client_name> --data '{
  "groups": ["<group_name>"]
}'
```

The `<client_name>` must be unique. If it is not, the certificate fingerprint of the client can be used.

For OIDC identities, run:

```
lxc query --request PATCH /1.0/auth/identities/oidc/<client_name> --data '{
  "groups": ["<group_name>"]
}'
```

The `<client_name>` must be unique. If it is not, the email address of the client can be used.

Confine users to specific LXD projects via Unix socket

If you use the LXD snap[101], you can configure the multi-user LXD daemon contained in the snap to dynamically create projects for all users in a specific user group.

To do so, set the `daemon.user.group` configuration option to the corresponding user group:

```
sudo snap set lxd daemon.user.group=<user_group>
```

Make sure that all user accounts that you want to be able to use LXD are a member of this group.

Once a member of the group issues a LXD command, LXD creates a confined project for this user and switches to this project. If LXD has not been *initialized* (page 35) at this point, it is automatically initialized (with the default settings).

If you want to customize the project settings, for example, to impose limits or restrictions, you can do so after the project has been created. To modify the project configuration, you must have full access to LXD, which means you must be part of the `lxd` group and not only the group that you configured as the LXD user group.

Related topics

Explanation:

- *Instances grouping with projects* (page 368)

Reference:

- *Project configuration* (page 509)

2.2.4. Storage

The following how-to guides cover common operations related to storage.

How to create, manage, and use storage:

How to manage storage pools

See the following sections for instructions on how to create, configure, view and resize *Storage pools* (page 350).

Create a storage pool

LXD creates a storage pool during initialization. You can add more storage pools later, using the same driver or different drivers.

CLI

UI

To create a storage pool, use the following command:

```
lxc storage create <pool_name> <driver> [configuration_options...]
```

See the *Storage drivers* (page 521) documentation for a list of available configuration options for each driver.

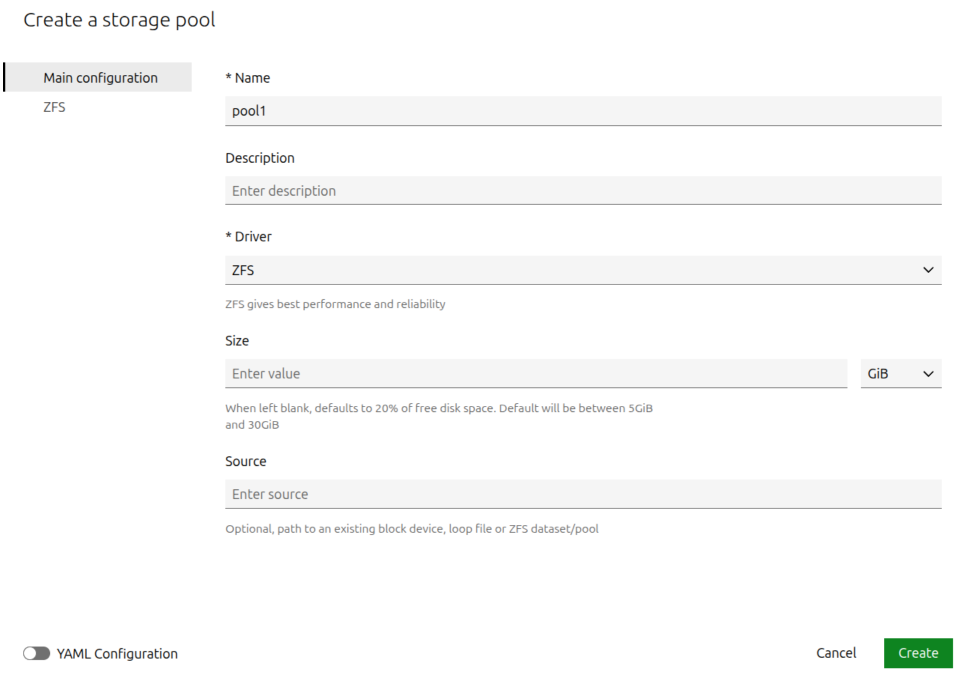
[101] https://snapcraft.io/lxd

To create a storage pool, select *Pools* from the *Storage* section of the main navigation.

On the resulting screen, click *Create pool* in the upper right corner.

From this screen, you can configure the name and description of your storage pool. You can select a storage driver from the *Driver* dropdown. Additional settings might appear, depending on the storage driver selected.

Click *Create* to create the storage pool.



By default, LXD sets up loop-based storage with a sensible default size/quota: 20% of the free disk space, with a minimum of 5 GiB and a maximum of 30 GiB.

Examples

CLI

UI

The following examples demonstrate how to create a storage pool using different types of storage drivers.

Create a directory pool

Create a directory pool named `pool1`:

```
lxc storage create pool1 dir
```

Use the existing directory `/data/lxd` for `pool2`:

```
lxc storage create pool2 dir source=/data/lxd
```

Create a Btrfs pool

Create a loop-backed pool named `pool1`:

```
lxc storage create pool1 btrfs
```

Use the existing Btrfs file system at `/some/path` for `pool2`:

```
lxc storage create pool2 btrfs source=/some/path
```

Create a pool named `pool3` on `/dev/sdX`:

```
lxc storage create pool3 btrfs source=/dev/sdX
```

Create an LVM pool

Create a loop-backed pool named `pool1` (the LVM volume group will also be called `pool1`):

```
lxc storage create pool1 lvm
```

Use the existing LVM volume group called `my-pool` for `pool2`:

```
lxc storage create pool2 lvm source=my-pool
```

Use the existing LVM thin pool called `my-pool` in volume group `my-vg` for `pool3`:

```
lxc storage create pool3 lvm source=my-vg lvm.thinpool_name=my-pool
```

Create a pool named `pool4` on `/dev/sdX` (the LVM volume group will also be called `pool4`):

```
lxc storage create pool4 lvm source=/dev/sdX
```

Create a pool named `pool5` on `/dev/sdX` with the LVM volume group name `my-pool`:

```
lxc storage create pool5 lvm source=/dev/sdX lvm.vg_name=my-pool
```

Create a ZFS pool

Create a loop-backed pool named `pool1` (the ZFS zpool will also be called `pool1`):

```
lxc storage create pool1 zfs
```

Create a loop-backed pool named `pool2` with the ZFS zpool name `my-tank`:

```
lxc storage create pool2 zfs zfs.pool_name=my-tank
```

Use the existing ZFS zpool `my-tank` for `pool3`:

```
lxc storage create pool3 zfs source=my-tank
```

Use the existing ZFS dataset `my-tank/slice` for `pool4`:

```
lxc storage create pool4 zfs source=my-tank/slice
```

Use the existing ZFS dataset `my-tank/zvol` for `pool5` and configure it to use ZFS block mode:

```
lxc storage create pool5 zfs source=my-tank/zvol volume.zfs.block_mode=yes
```

Create a pool named `pool6` on `/dev/sdX` (the ZFS zpool will also be called `pool6`):

```
lxc storage create pool6 zfs source=/dev/sdX
```

Create a pool named `pool7` on `/dev/sdX` with the ZFS zpool name `my-tank`:

```
lxc storage create pool7 zfs source=/dev/sdX zfs.pool_name=my-tank
```

Create a Ceph RBD pool

Create an OSD storage pool named `pool1` in the default Ceph cluster (named `ceph`):

```
lxc storage create pool1 ceph
```

Create an OSD storage pool named `pool2` in the Ceph cluster `my-cluster`:

```
lxc storage create pool2 ceph ceph.cluster_name=my-cluster
```

Create an OSD storage pool named `pool3` with the on-disk name `my-osd` in the default Ceph cluster:

```
lxc storage create pool3 ceph ceph.osd.pool_name=my-osd
```

Use the existing OSD storage pool `my-already-existing-osd` for `pool4`:

```
lxc storage create pool4 ceph source=my-already-existing-osd
```

Use the existing OSD erasure-coded pool `ecpool` and the OSD replicated pool `rpl-pool` for `pool5`:

```
lxc storage create pool5 ceph source=rpl-pool ceph.osd.data_pool_name=ecpool
```

Create a CephFS pool

> **Note**
>
> Each CephFS file system consists of two OSD storage pools, one for the actual data and one for the file metadata.

Use the existing CephFS file system `my-filesystem` for `pool1`:

```
lxc storage create pool1 cephfs source=my-filesystem
```

Use the sub-directory `my-directory` from the `my-filesystem` file system for `pool2`:

```
lxc storage create pool2 cephfs source=my-filesystem/my-directory
```

Create a CephFS file system `my-filesystem` with a data pool called `my-data` and a metadata pool called `my-metadata` for `pool3`:

```
lxc storage create pool3 cephfs source=my-filesystem cephfs.create_missing=true
cephfs.data_pool=my-data cephfs.meta_pool=my-metadata
```

Create a Ceph Object pool

> **Note**
>
> When using the Ceph Object driver, you must have a running Ceph Object Gateway `radosgw`[102] URL available beforehand.

Use the existing Ceph Object Gateway `https://www.example.com/radosgw` to create `pool1`:

```
lxc storage create pool1 cephobject cephobject.radosgw.endpoint=https://www.
example.com/radosgw
```

Create a Dell PowerFlex pool

Create a storage pool named `pool1` using the PowerFlex pool `sp1` in the protection domain `pd1`:

```
lxc storage create pool1 powerflex powerflex.pool=sp1 powerflex.domain=pd1
powerflex.gateway=https://powerflex powerflex.user.name=lxd powerflex.user.
password=foo
```

Create a storage pool named `pool2` using the ID of PowerFlex pool `sp1`:

```
lxc storage create pool2 powerflex powerflex.pool=<ID of sp1> powerflex.
gateway=https://powerflex powerflex.user.name=lxd powerflex.user.password=foo
```

Create a storage pool named `pool3` that uses PowerFlex volume snapshots (see *Limitations* (page 542)) when creating volume copies:

```
lxc storage create pool3 powerflex powerflex.clone_copy=false powerflex.pool=<id
of sp1> powerflex.gateway=https://powerflex powerflex.user.name=lxd powerflex.
user.password=foo
```

Create a storage pool named `pool4` that uses a PowerFlex gateway with a certificate that is not trusted:

```
lxc storage create pool4 powerflex powerflex.gateway.verify=false powerflex.pool=
<id of sp1> powerflex.gateway=https://powerflex powerflex.user.name=lxd powerflex.
user.password=foo
```

Create a storage pool named `pool5` that explicitly uses the PowerFlex SDC:

```
lxc storage create pool5 powerflex powerflex.mode=sdc powerflex.pool=<id of sp1>
powerflex.gateway=https://powerflex powerflex.user.name=lxd powerflex.user.
password=foo
```

[102] https://docs.ceph.com/en/latest/radosgw/

Create a Pure Storage pool

Create a storage pool named `pool1` that uses NVMe/TCP by default:

```
lxc storage create pool1 pure pure.gateway=https://<pure-storage-address> pure.
api.token=<pure-storage-api-token>
```

Create a storage pool named `pool2` that uses a Pure Storage gateway with a certificate that is not trusted:

```
lxc storage create pool2 pure pure.gateway=https://<pure-storage-address> pure.
gateway.verify=false pure.api.token=<pure-storage-api-token>
```

Create a storage pool named `pool3` that uses iSCSI to connect to Pure Storage array:

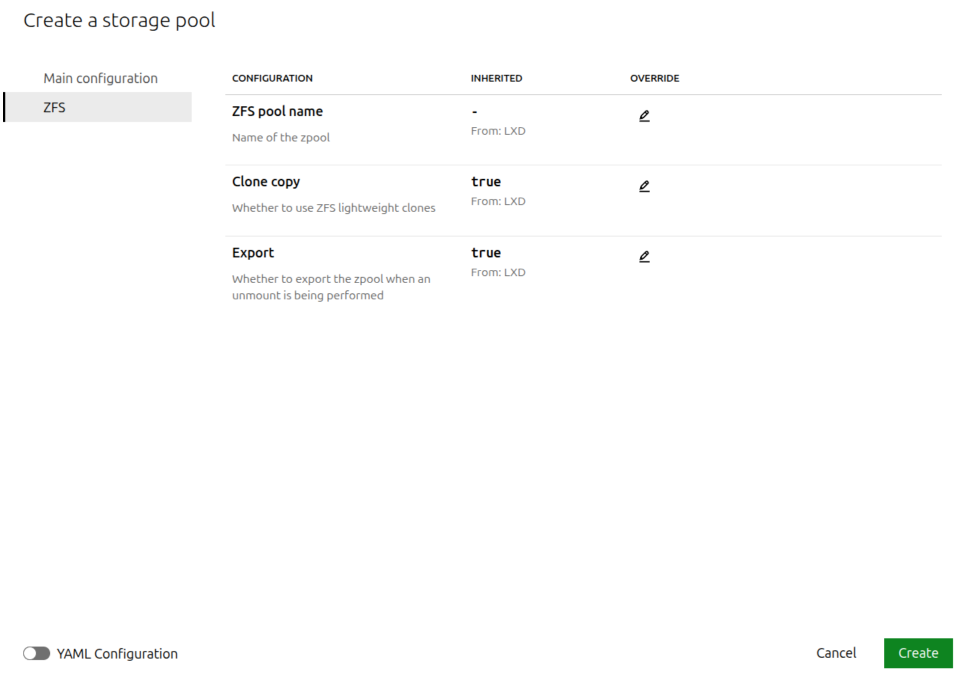
```
lxc storage create pool3 pure pure.gateway=https://<pure-storage-address> pure.
api.token=<pure-storage-api-token> pure.mode=iscsi
```

Create a storage pool named `pool4` that uses NVMe/TCP to connect to Pure Storage array via specific target addresses:

```
lxc storage create pool4 pure pure.gateway=https://<pure-storage-address> pure.
api.token=<pure-storage-api-token> pure.mode=nvme pure.target=<target_address_1>,
<target_address_2>
```

You can select a storage driver from the *Driver* dropdown.

Some storage drivers offer additional settings. Click the driver name in the secondary menu to further configure the storage pool.



See the *Storage drivers* (page 521) documentation for a list of available configuration options for each driver.

Create a storage pool in a cluster

If you are running a LXD cluster and want to add a storage pool, you must create the storage pool for each cluster member separately. The reason for this is that the configuration, for example, the storage location or the size of the pool, might be different between cluster members.

CLI

UI

To create a storage pool via the CLI, start by creating a pending storage pool on each member with the `--target=<cluster_member>` flag and the appropriate configuration for the member.

Make sure to use the same storage pool name for all members. Then create the storage pool without specifying the `--target` flag to actually set it up.

Also see *How to configure storage for a cluster* (page 290).

> **Note**
>
> For most storage drivers, the storage pools exist locally on each cluster member. That means that if you create a storage volume in a storage pool on one member, it will not be available on other cluster members.
>
> This behavior is different for Ceph-based storage pools (`ceph`, `cephfs` and `cephobject`) where each storage pool exists in one central location and therefore, all cluster members access the same storage pool with the same storage volumes.

Examples

See the following examples for different storage drivers for instructions on how to create local or remote storage pools in a cluster.

Create a local storage pool

Create a storage pool named `my-pool` using the ZFS driver at different locations and with different sizes on three cluster members:

```
~$ lxc storage create my-pool zfs source=/dev/sdX size=10GiB --target=vm01
Storage pool my-pool pending on member vm01 ~$ lxc storage create my-pool zfs
source=/dev/sdX size=15GiB --target=vm02 Storage pool my-pool pending on
member vm02 ~$ lxc storage create my-pool zfs source=/dev/sdY size=10GiB
--target=vm03 Storage pool my-pool pending on member vm03 ~$ lxc storage
create my-pool zfs Storage pool my-pool created
```

Create a remote storage pool

Create a storage pool named `my-remote-pool` using the Ceph RBD driver and the on-disk name `my-osd` on three cluster members. Because the *`ceph.osd.pool_name`* (page 536) configuration setting isn't member-specific, it must be set when creating the actual storage pool:

```
~$ lxc storage create my-remote-pool ceph --target=vm01 Storage pool
my-remote-pool pending on member vm01 ~$ lxc storage create my-remote-pool
ceph --target=vm02 Storage pool my-remote-pool pending on member vm02 ~$ lxc
storage create my-remote-pool ceph --target=vm03 Storage pool my-remote-pool
pending on member vm03 ~$ lxc storage create my-remote-pool ceph
ceph.osd.pool_name=my-osd Storage pool my-remote-pool created
```

Create a second storage pool named `my-remote-pool2` using the Dell PowerFlex driver in SDC mode and the pool `sp1` in protection domain `pd1`:

```
~$ lxc storage create my-remote-pool2 powerflex --target=vm01 Storage pool
my-remote-pool2 pending on member vm01 ~$ lxc storage create my-remote-pool2
powerflex --target=vm02 Storage pool my-remote-pool2 pending on member vm02 ~$
lxc storage create my-remote-pool2 powerflex --target=vm03 Storage pool
my-remote-pool2 pending on member vm03 ~$ lxc storage create my-remote-pool2
powerflex powerflex.mode=sdc powerflex.pool=sp1 powerflex.domain=pd1
powerflex.gateway=https://powerflex powerflex.user.name=lxd
powerflex.user.password=foo Storage pool my-remote-pool2 created
```

Create a third storage pool named `my-remote-pool3` using the Pure Storage driver:

```
~$ lxc storage create my-remote-pool3 pure --target=vm01 Storage pool
my-remote-pool3 pending on member vm01 ~$ lxc storage create my-remote-pool3
pure --target=vm02 Storage pool my-remote-pool3 pending on member vm02 ~$ lxc
storage create my-remote-pool3 pure --target=vm03 Storage pool
my-remote-pool3 pending on member vm03 ~$ lxc storage create my-remote-pool3
pure pure.gateway=https://<pure-storage-address>
pure.api.token=<pure-storage-api-token> Storage pool my-remote-pool3 created
```

To create a storage pool in a cluster, select *Pools* from the *Storage* section of the main navigation, then click *Create pool* in the upper right corner.

On the resulting page, configure the storage pool's name and description. Depending on the selected driver, some settings can be configured per cluster member or applied globally to the cluster.

Finally, click *Create* to create the storage pool.

Configure storage pool settings

See the *Storage drivers* (page 521) documentation for the available configuration options for each storage driver.

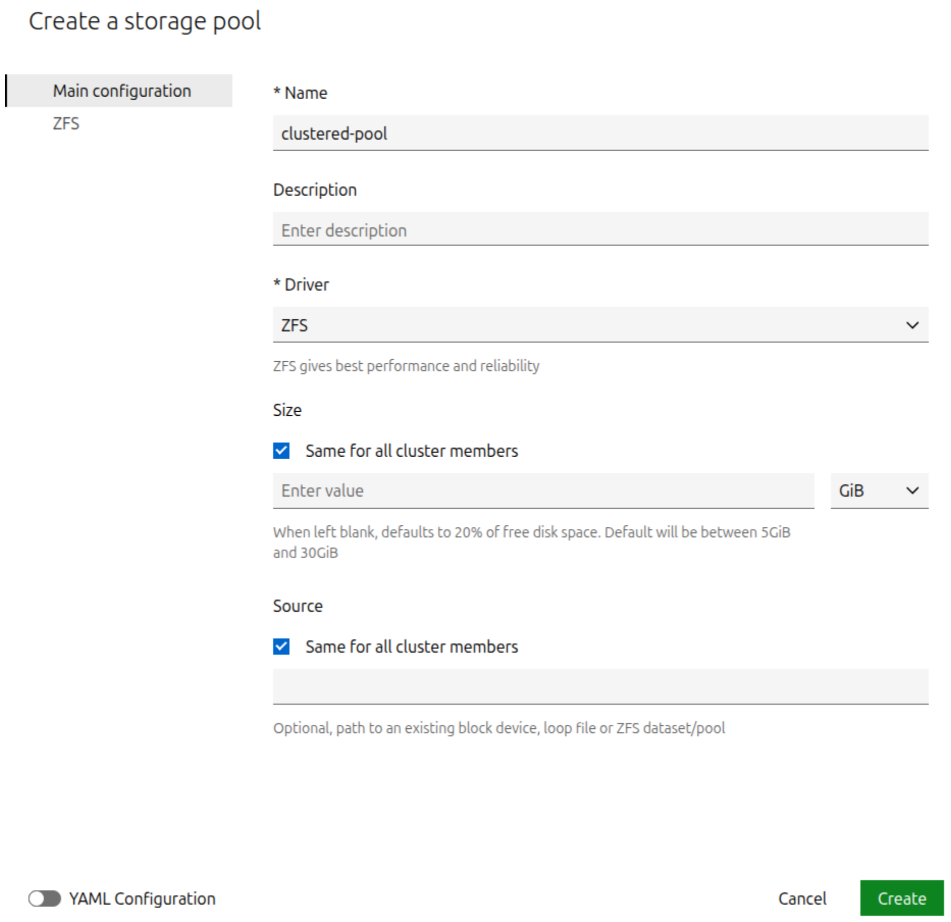
General keys for a storage pool (like `source`) are top-level. Driver-specific keys are namespaced by the driver name.

CLI

UI

Use the following command to set configuration options for a storage pool:

```
lxc storage set <pool_name> <key> <value>
```



For example, to turn off compression during storage pool migration for a `dir` storage pool, use the following command:

```
lxc storage set my-dir-pool rsync.compression false
```

You can also edit the storage pool configuration by using the following command:

```
lxc storage edit <pool_name>
```

To configure a storage pool, select *Pools* from the *Storage* section of the Main navigation.

The resulting screen shows a list of existing storage pools. Click a pool's name to access its details.

Go to the *Configuration* tab. Here, you can configure settings such as the storage pool description.

After making changes, click the *Save changes* button. This button also displays the number of changes you have made.

View storage pools

You can display a list of all available storage pools and check their configuration.

CLI

UI

Use the following command to list all available storage pools:

```
lxc storage list
```

The resulting table contains the storage pool that you created during initialization (usually called `default` or `local`) and any storage pools that you added.

To show detailed information about a specific pool, use the following command:

```
lxc storage show <pool_name>
```

To see usage information for a specific pool, run the following command:

```
lxc storage info <pool_name>
```

To view available storage pools in the UI, select *Pools* from the *Storage* section of the main navigation.

Resize a storage pool

If you need more storage, you can increase the size (quota) of your storage pool. You can only grow the pool (increase its size), not shrink it.

CLI

UI

In the CLI, resize a storage pool by changing the `size` configuration key:

```
lxc storage set <pool_name> size=<new_size>
```

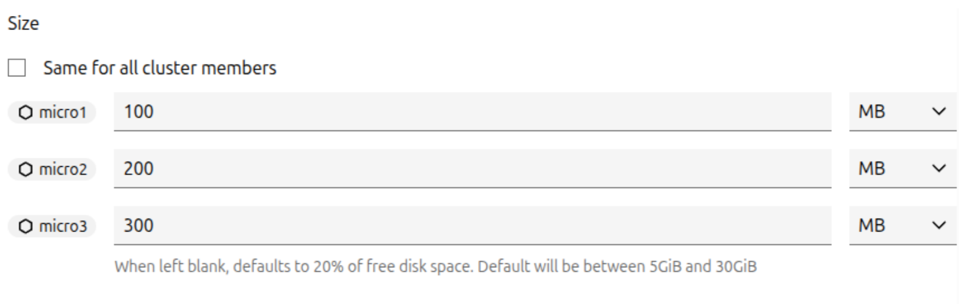
This will only work for loop-backed storage pools that are managed by LXD.

To resize a storage pool in the UI, select *Pools* from the *Storage* section of the main navigation.

Click the name of a storage pool to open its details page, then go to its *Configuration* tab. Edit the *Size* field.

After making changes, click the *Save changes* button. This button also displays the number of changes you have made.

In clustered environments, the *Size* field appears as a per-member selector, allowing you to configure the size for each cluster member.



How to manage storage volumes

See the following sections for instructions on how to create, configure, view and resize *Storage volumes* (page 352).

View storage volumes

You can display a list of all available storage volumes and check their configuration.

CLI

UI

To list all available storage volumes, use the following command:

```
lxc storage volume list
```

To display the storage volumes for all projects (not only the default project), add the `--all-projects` flag.

You can also display the storage volumes in a specific storage pool:

```
lxc storage volume list my-pool
```

The resulting table contains, among other information, the *storage volume type* (page 352) and the *content type* (page 352) for each storage volume.

> **Note**
>
> Custom storage volumes can use the same name as instance volumes. For example, you might have a container named `c1` with a container storage volume named `c1` and a custom storage volume named `c1`. Therefore, to distinguish between instance storage volumes and custom storage volumes, all instance storage volumes must be referred to as <volume_type>/<volume_name> (for example, `container/c1` or `virtual-machine/vm`) in commands.

To show detailed configuration information about a specific volume, use the following command:

```
lxc storage volume show my-pool custom/my-volume
```

To show state information about a specific volume, use the following command:

```
lxc storage volume info my-pool virtual-machine/my-vm
```

In both commands, the default *storage volume type* (page 352) is `custom`, so you can leave out the `custom/` when displaying information about a custom storage volume.

From the main navigation, select *Storage* > *Volumes*. The resulting page displays a table of available volumes. You can sort volumes by their pool by clicking the *Pool* column header of the table.

Create a custom storage volume

When you create an instance, LXD automatically creates a storage volume that is used as the root disk for the instance.

You can add custom storage volumes to your instances. Such custom storage volumes are independent of the instance, which means that they can be backed up separately and are retained until you delete them. Custom storage volumes with content type `filesystem` can also be shared between different instances.

See *Storage volumes* (page 352) for detailed information.

Create the volume

CLI

UI

Use the following command to create a custom storage volume `vol1` of type `filesystem` in storage pool `my-pool`:

```
lxc storage volume create my-pool vol1
```

By default, custom storage volumes use the `filesystem` *content type* (page 352). To create a custom volume with content type `block`, add the `--type` flag:

```
lxc storage volume create my-pool vol2 --type=block
```

From the main navigation, select *Storage* > *Volumes*.

On the resulting page, click *Create volume* in the upper-right corner.

You can then configure the name and size of your storage volume.

You can select a content type from the *Content type* dropdown. Additional settings might appear, depending on the content type selected.

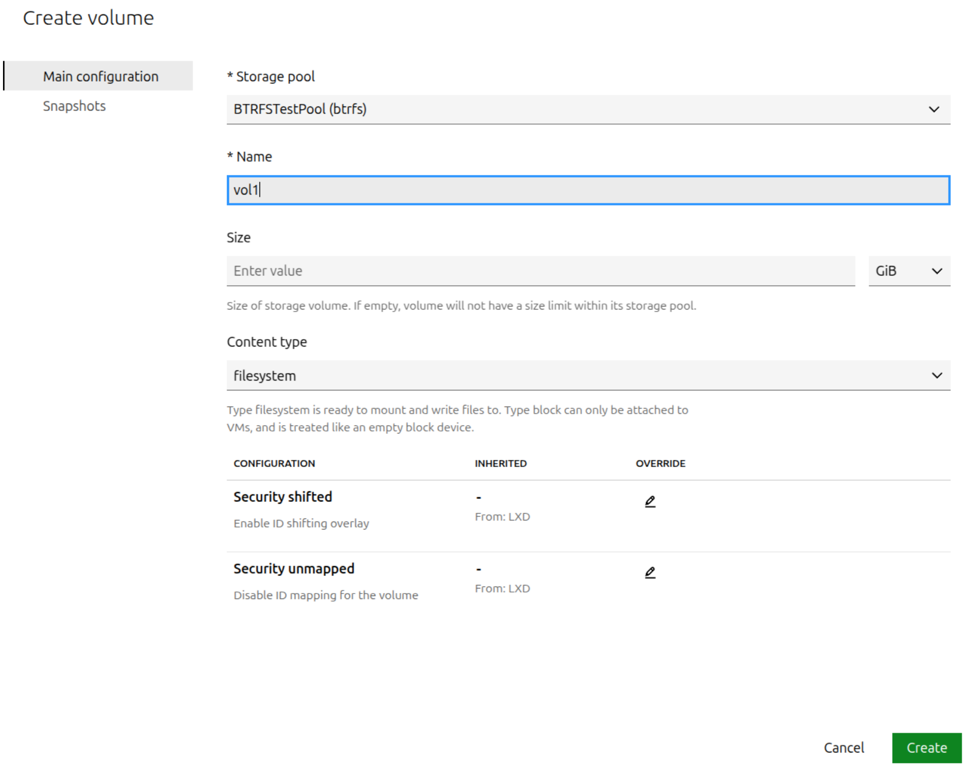
Click *Create* to create the storage pool.

Attach the volume to an instance

After creating a custom storage volume, you can add it to one or more instances as a *disk device* (page 478).

The following restrictions apply:

- Storage volumes of *content type* (page 352) `block` or `iso` cannot be attached to containers, only to virtual machines.
- Storage volumes of *content type* (page 352) `block` that don't have `security.shared` enabled cannot be attached to more than one instance at the same time. Attaching a `block` volume to more than one instance at a time risks data corruption.
- Storage volumes of *content type* (page 352) `iso` are always read-only, and can therefore be attached to more than one virtual machine at a time without corrupting data.
- Storage volumes of *content type* (page 352) `filesystem` can't be attached to virtual machines while they're running.



- You cannot attach a storage volume from a local storage pool (a pool that uses the *Directory* (page 553), *Btrfs* (page 521), *ZFS* (page 563), or *LVM* (page 557) driver) to an instance that has *`migration.stateful`* (page 429) set to `true`. You must set *`migration.stateful`* (page 429) to `false` on the instance. Note that doing so makes the instance ineligible for *live migration* (page 136).

CLI

UI

Use the following command to attach a custom storage volume `fs-vol` with content type `filesystem` to instance `c1`. `/data` is the mount point for the storage volume inside the instance:

```
lxc storage volume attach my-pool fs-vol c1 /data
```

Custom storage volumes with the content type `block` do not take a mount point:

```
lxc storage volume attach my-pool bl-vol vm1
```

By default, custom storage volumes are added to the instance with the volume name as the *device* (page 447) name. If you want to use a different device name, you can add it to the command:

```
lxc storage volume attach my-pool fs-vol c1 filesystem-volume /data
lxc storage volume attach my-pool bl-vol vm1 block-volume
```

Attach the volume as a device

The *lxc storage volume attach* (page 898) command is a shortcut for adding a disk device to an instance. The following commands have the same effect as the corresponding commands above:

```
lxc config device add c1 filesystem-volume disk pool=my-pool source=fs-vol path=/
data
lxc config device add vm1 block-volume disk pool=my-pool source=bl-vol
```

This allows adding further configuration for the device. See *disk device* (page 478) for all available device options.

Configure I/O options

When you attach a storage volume to an instance as a *disk device* (page 478), you can configure I/O limits for it. To do so, set the *limits.read* (page 482), *limits.write* (page 482) or *limits.max* (page 482) options to the corresponding limits. See the *Type: disk* (page 478) reference for more information.

The limits are applied through the Linux `blkio` cgroup controller, which makes it possible to restrict I/O at the disk level (but nothing finer grained than that).

Note

Because the limits apply to a whole physical disk rather than a partition or path, the following restrictions apply:

- Limits will not apply to file systems that are backed by virtual devices (for example, device mapper).
- If a file system is backed by multiple block devices, each device will get the same limit.
- If two disk devices that are backed by the same disk are attached to the same instance, the limits of the two devices will be averaged.

All I/O limits only apply to actual block device access. Therefore, consider the file system's own overhead when setting limits. Access to cached data is not affected by the limit.

For VMs the way the disk is exposed to the guest and its behavior can be configured. To do so, set the *io.bus* (page 481), *io.cache* (page 481) or *io.threads* (page 481) options. See the *Type: disk* (page 478) reference for more information.

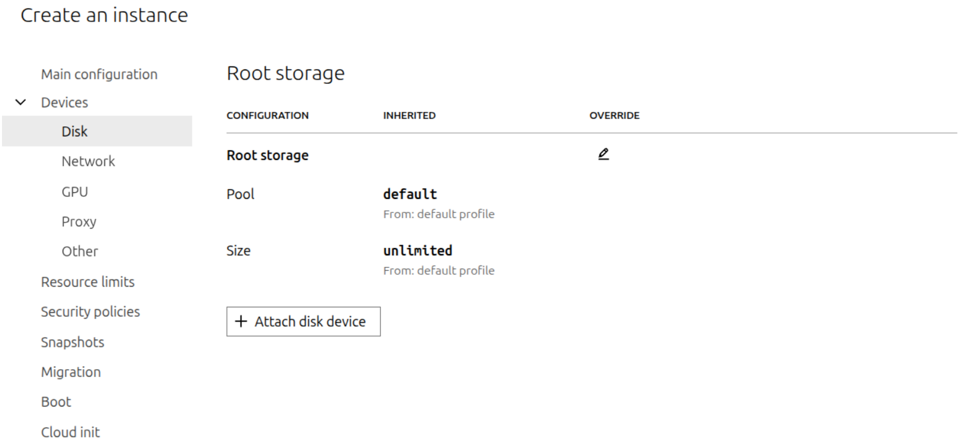
You can attach a storage volume to an existing instance, or when creating a new instance:

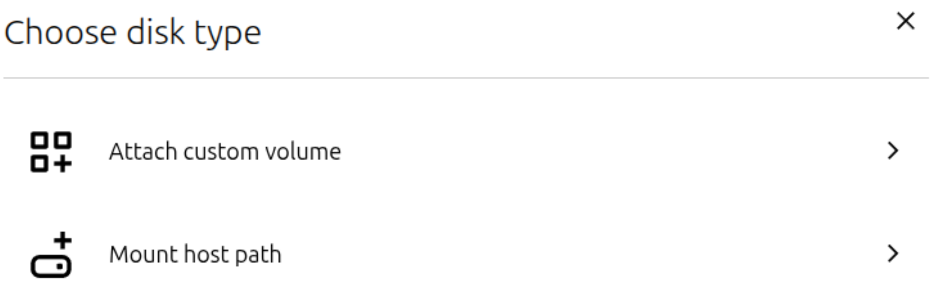
- For an existing instance, select *Instances* from the main navigation, then select the target instance to view its details page. Open its *Configuration* tab.
- For a new instance, you must first select a base image during the instance creation process.

In either scenario, then select *Disk* from the *Devices* section of the secondary menu.

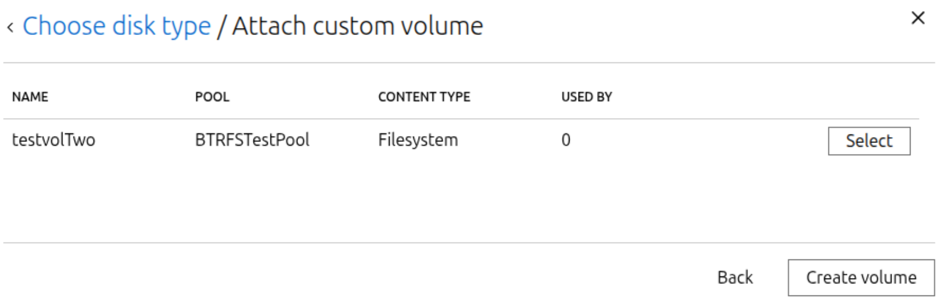
Click *Attach disk device*.

The resulting modal allows you to choose your disk type. Select *Attach custom volume*:





Next, you can either select a pre-existing volume to attach to the instance by clicking its corresponding *Select* button, or create a new custom volume by clicking *Create volume*:



Once the modal closes, you might be required to add a mount point file path in the *Mount point* field. Finally, you can save your instance changes. If you are in the instance creation process, create your instance by clicking *Create*.

Use the volume for backups or images

Instead of attaching a custom volume to an instance as a disk device, you can also use it as a special kind of volume to store *backups* (page 316) or *images* (page 348).

CLI

UI

To do so, you must set the corresponding *server configuration* (page 411):

- To use a custom volume `my-backups-volume` to store the backup tarballs:

```
lxc config set storage.backups_volume=my-pool/my-backups-volume
```

- To use a custom volume `my-images-volume` to store the image tarballs:

```
lxc config set storage.images_volume=my-pool/my-images-volume
```

To use a volume to store backups or images, select *Settings* from the main navigation. From this page, set the value of the *storage.backups_volume* key or the *storage.images_volume* key to the name of the target storage volume, then select *Save*.

Configure storage volume settings

See the *Storage drivers* (page 521) documentation for a list of available storage volume configuration options for each driver.

CLI

UI

To set the maximum size of custom storage volume `my-volume` to 1 GiB, use the following command:

```
lxc storage volume set my-pool my-volume size=1GiB
```

The default *storage volume type* (page 352) is `custom`, but other volume types can be configured by using the `<volume_type>/<volume_name>` syntax.

To set the snapshot expiry time for virtual machine `my-vm` to one month, use the following command:

```
lxc storage volume set my-pool virtual-machine/my-vm snapshots.expiry=1M
```

You can also edit the storage volume configuration as YAML in a text editor:

```
lxc storage volume edit my-pool virtual-machine/my-vm
```

Configure default values for storage volumes

You can define default volume configurations for a storage pool. To do so, set a storage pool configuration with a `volume` prefix: `volume.<KEY>=<VALUE>`.

This value is used for all new storage volumes in the pool, unless it is explicitly overridden. In general, the defaults set at the storage pool level can be overridden through a volume's configuration. For storage volumes of *type* (page 352) `container` or `virtual-machine`, the pool's volume configuration can be overridden via the instance configuration.

For example, to set the default volume size for `my-pool`, use the following command:

```
lxc storage set my-pool volume.size=15GiB
```

Canonical

Attach instance root volumes to other instances

Virtual-machine root volumes can be attached as disk devices to other virtual machines. In order to prevent concurrent access, `security.protection.start` must be set on an instance before its root volume can be attached to another virtual-machine.

> **Caution**
>
> Because instances created from the same image share the same partition and file system UUIDs and labels, booting an instance with two root file systems mounted may result in the wrong root file system being used. This may result in unexpected behavior or data loss. **It is strongly recommended to only attach virtual-machine root volumes to other virtual machines when the target virtual-machine is running.**

Assuming `vm1` is stopped and `vm2` is running, attach the `virtual-machine/vm1` storage volume to `vm2`:

```
lxc config set vm1 security.protection.start=true
lxc storage volume attach my-pool virtual-machine/vm1 vm2
```

`virtual-machine/vm1` must be detached from `vm2` before `security.protection.start` can be unset from `vm1`:

```
lxc storage volume detach my-pool virtual-machine/vm1 vm2
lxc config unset vm1 security.protection.start
```

`security.shared` can also be used on `virtual-machine` volumes to enable concurrent access. Note that concurrent access to block volumes may result in data loss.

Attaching virtual machine snapshots to other instances

Virtual-machine snapshots can also be attached to instances with the *`source.snapshot`* (page 484) disk device configuration key.

```
lxc config device add v1 v2-root-snap0 disk pool=my-pool source=vm2 source.
type=virtual-machine source.snapshot=snap0
```

Resize a storage volume

If you need more storage in a volume, you can increase the size of your storage volume. In some cases, it is also possible to reduce the size of a storage volume.

To adjust a storage volume's quota, set its `size` configuration. For example, to resize `my-volume` in storage pool `my-pool` to `15GiB`, use the following command:

```
lxc storage volume set my-pool my-volume size=15GiB
```

> **Important**
>
> - Growing a volume is possible if the storage pool has sufficient storage.

- Shrinking a storage volume is only possible for storage volumes with content type `filesystem`. It is not guaranteed to work though, because you cannot shrink storage below its current used size.
- Shrinking a storage volume with content type `block` is not possible.

To configure a custom storage volume, select *Storage* > *Volumes* from the main navigation. Next, click the name of your target storage volume to view its details page.

Note

Volume details pages are only available for volumes of type Custom. Volumes of other types—such as Instance root disks—can also be accessed from the Volumes page and redirect to their respective entity overview or list page.

To sort the Volumes table by type, you can click the *Content type* column header.

On the volume's overview page, go to the *Configuration* tab. Here, you can configure settings such as the storage volume size. Further configuration options can be found in the secondary menu. After making changes, click the *Save changes* button. This button also displays the number of changes you have made.

Create a storage volume in a cluster

For most storage drivers, custom storage volumes are not replicated across the cluster and exist only on the member for which they were created. This behavior differs for remote storage pools (`ceph`, `cephfs` and `powerflex`), where volumes are available from any cluster member.

CLI

UI

To add a custom storage volume on a cluster member, add the `--target` flag:

```
lxc storage volume create <pool-name> <volume-name> --target=<member-name>
```

Example:

```
lxc storage volume create my-pool my-volume --target=my-member
```

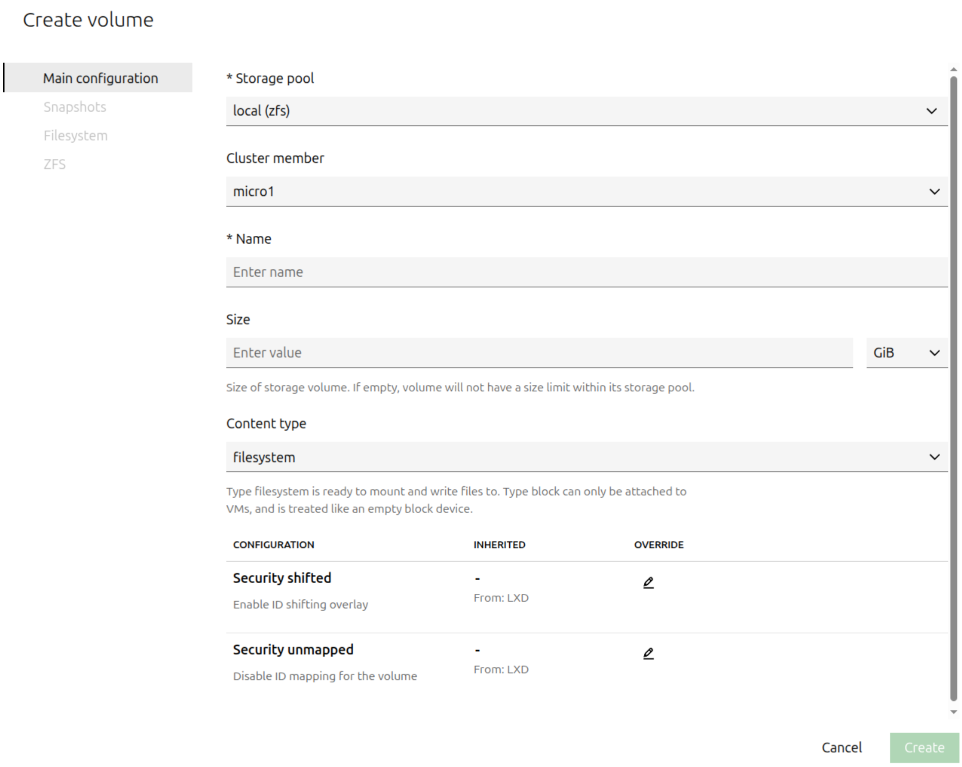
To create a custom storage volume of type `iso`, use `import` instead of `create`:

```
lxc storage volume import <pool-name> <path-to-iso> <volume-name> --type=iso
```

To create a storage volume in a clustered environment, select *Storage* > *Volumes* from the main navigation. On the Volumes page, click *Create volume* in the upper-right corner.

On the volume creation page, select the cluster member on which to base the storage volume from the *Cluster member* dropdown. This dropdown is only available if the storage pool selected for this volume is cluster-member specific, rather than shared across the cluster.

To find out more about clusters in LXD, see:



- *Clustering how-to guides* (page 280)
- *An explanation about clusters* (page 370)

How to manage storage buckets and keys

See the following sections for instructions on how to create, configure, view and resize *Storage buckets* (page 353) and how to manage storage bucket keys.

Install requirements for local storage buckets

LXD uses MinIO[103] to set up local storage buckets. To use this feature with LXD, you must install both the server and client binaries.

- MinIO Server:
  - Source:
    - MinIO Server on GitHub[104]
  - Direct download for various architectures:
    - MinIO Server pre-built for `amd64`[105]
    - MinIO Server pre-built for `arm64`[106]
    - MinIO Server pre-built for `arm`[107]

[103] https://min.io
[104] https://github.com/minio/minio
[105] https://dl.min.io/server/minio/release/linux-amd64/minio
[106] https://dl.min.io/server/minio/release/linux-arm64/minio
[107] https://dl.min.io/server/minio/release/linux-arm/minio

* MinIO Server pre-built for `ppc64le`[108]
* MinIO Server pre-built for `s390x`[109]

- MinIO Client:
  - Source:
    * MinIO Client on GitHub[110]
  - Direct download for various architectures:
    * MinIO Client pre-built for `amd64`[111]
    * MinIO Client pre-built for `arm64`[112]
    * MinIO Client pre-built for `arm`[113]
    * MinIO Client pre-built for `ppc64le`[114]
    * MinIO Client pre-built for `s390x`[115]

If LXD is installed from a Snap, you must configure the snap environment to detect the binaries, and restart LXD. Note that the path to the directory containing the binaries must not be under the home directory of any user.

```
snap set lxd minio.path=/path/to/directory/containing/both/binaries
snap restart lxd
```

If LXD is installed from another source, both binaries must be included in the `$PATH` that LXD was started with.

Configure the S3 address

If you want to use storage buckets on local storage (thus in a `dir`, `btrfs`, `lvm`, or `zfs` pool), you must configure the S3 address for your LXD server. This is the address that you can then use to access the buckets through the S3 protocol.

To configure the S3 address, set the *`core.storage_buckets_address`* (page 404) server configuration option. For example:

```
lxc config set core.storage_buckets_address :8555
```

Manage storage buckets

Storage buckets provide access to object storage exposed using the S3 protocol.

Unlike custom storage volumes, storage buckets are not added to an instance, but applications can instead access them directly via their URL.

See *Storage buckets* (page 353) for detailed information.

[108] https://dl.min.io/server/minio/release/linux-ppc64le/minio
[109] https://dl.min.io/server/minio/release/linux-s390x/minio
[110] https://github.com/minio/mc
[111] https://dl.min.io/client/mc/release/linux-amd64/mc
[112] https://dl.min.io/client/mc/release/linux-arm64/mc
[113] https://dl.min.io/client/mc/release/linux-arm/mc
[114] https://dl.min.io/client/mc/release/linux-ppc64le/mc
[115] https://dl.min.io/client/mc/release/linux-s390x/mc

Create a storage bucket

Use the following command to create a storage bucket in a storage pool:

```
lxc storage bucket create <pool_name> <bucket_name> [configuration_options...]
```

See the *Storage drivers* (page 521) documentation for a list of available storage bucket configuration options for each driver that supports object storage.

To add a storage bucket on a cluster member, add the `--target` flag:

```
lxc storage bucket create <pool_name> <bucket_name> --target=<cluster_member>
[configuration_options...]
```

Note

For most storage drivers, storage buckets are not replicated across the cluster and exist only on the member for which they were created. This behavior is different for `cephobject` storage pools, where buckets are available from any cluster member.

Configure storage bucket settings

See the *Storage drivers* (page 521) documentation for the available configuration options for each storage driver that supports object storage.

Use the following command to set configuration options for a storage bucket:

```
lxc storage bucket set <pool_name> <bucket_name> <key> <value>
```

For example, to set the size (quota) of a bucket, use the following command:

```
lxc storage bucket set my-pool my-bucket size 1MiB
```

You can also edit the storage bucket configuration by using the following command:

```
lxc storage bucket edit <pool_name> <bucket_name>
```

Use the following command to delete a storage bucket and its keys:

```
lxc storage bucket delete <pool_name> <bucket_name>
```

View storage buckets

You can display a list of all available storage buckets in a storage pool and check their configuration.

To list all available storage buckets in a storage pool, use the following command:

```
lxc storage bucket list <pool_name>
```

To show detailed information about a specific bucket, use the following command:

```
lxc storage bucket show <pool_name> <bucket_name>
```

Resize a storage bucket

By default, storage buckets do not have a quota applied.

To set or change a quota for a storage bucket, set its size configuration:

```
lxc storage bucket set <pool_name> <bucket_name> size <new_size>
```

> **Important**
>
> - Growing a storage bucket usually works (if the storage pool has sufficient storage).
> - You cannot shrink a storage bucket below its current used size.

Manage storage bucket keys

To access a storage bucket, applications must use a set of S3 credentials made up of an *access key* and a *secret key*. You can create multiple sets of credentials for a specific bucket.

Each set of credentials is given a key name. The key name is used only for reference and does not need to be provided to the application that uses the credentials.

Each set of credentials has a *role* that specifies what operations they can perform on the bucket.

The roles available are:

- `admin` - Full access to the bucket
- `read-only` - Read-only access to the bucket (list and get files only)

If the role is not specified when creating a bucket key, the role used is `read-only`.

Create storage bucket keys

Use the following command to create a set of credentials for a storage bucket:

```
lxc storage bucket key create <pool_name> <bucket_name> <key_name> [configuration_options...]
```

Use the following command to create a set of credentials for a storage bucket with a specific role:

```
lxc storage bucket key create <pool_name> <bucket_name> <key_name> --role=admin [configuration_options...]
```

These commands will generate and display a random set of credential keys.

Edit or delete storage bucket keys

Use the following command to edit an existing bucket key:

```
lxc storage bucket key edit <pool_name> <bucket_name> <key_name>
```

Use the following command to delete an existing bucket key:

```
lxc storage bucket key delete <pool_name> <bucket_name> <key_name>
```

View storage bucket keys

Use the following command to see the keys defined for an existing bucket:

```
lxc storage bucket key list <pool_name> <bucket_name>
```

Use the following command to see a specific bucket key:

```
lxc storage bucket key show <pool_name> <bucket_name> <key_name>
```

How to create an instance in a specific storage pool

Instance storage volumes are created in the storage pool that is specified by the instance's root disk device. This configuration is normally provided by the profile or profiles applied to the instance. See *Default storage pool* (page 351) for detailed information.

CLI

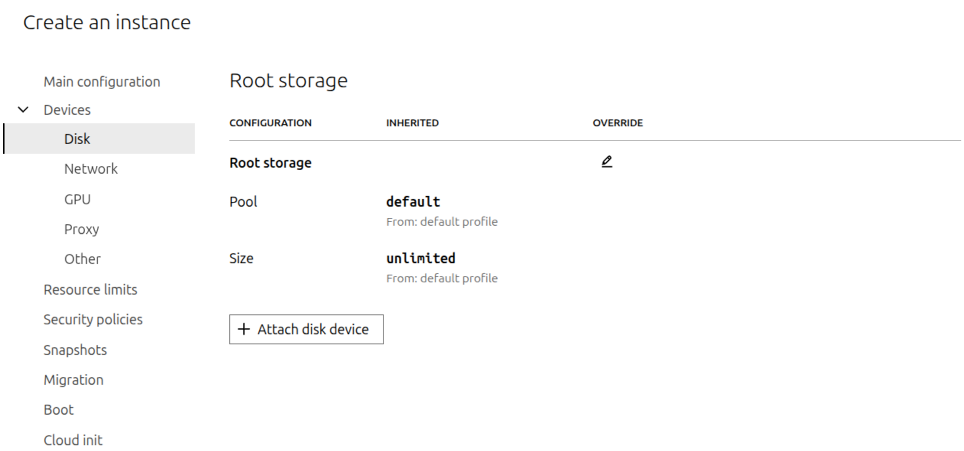
UI

To use a different storage pool when creating or launching an instance, add the `--storage` flag. This flag overrides the root disk device from the profile. For example:

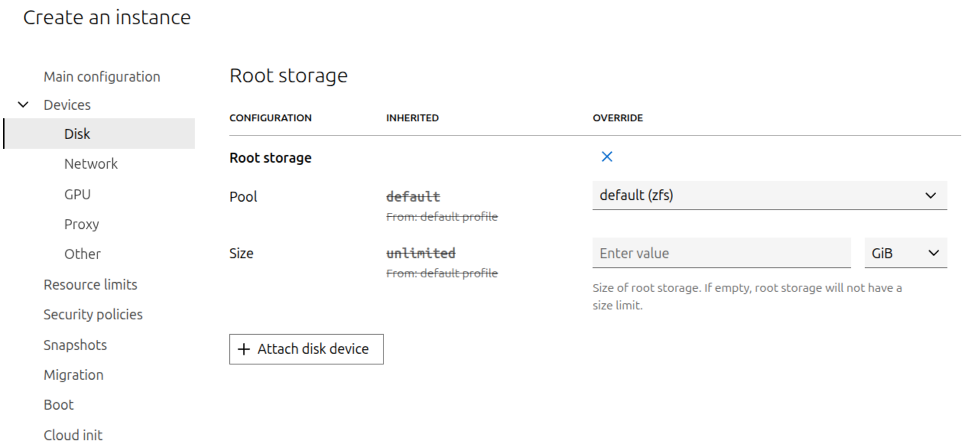
```
lxc launch <image> <instance_name> --storage <storage_pool>
```

To create an instance in a specific storage pool, override the root storage during instance creation.

To do this, begin the *instance creation wizard* (page 73). Once the *Base Image* is selected, the *Devices* section of the left-hand sub-menu becomes available. From this section, select *Devices > Disk*.



In this page, in the *Override* column, click the Edit button to create an override.



From here, you can override the pool and size of the root storage by editing their respective fields.

Move instance storage volumes to another pool

To move an instance storage volume to another storage pool, *stop the instance* (page 94) that contains the storage volume you want to move.

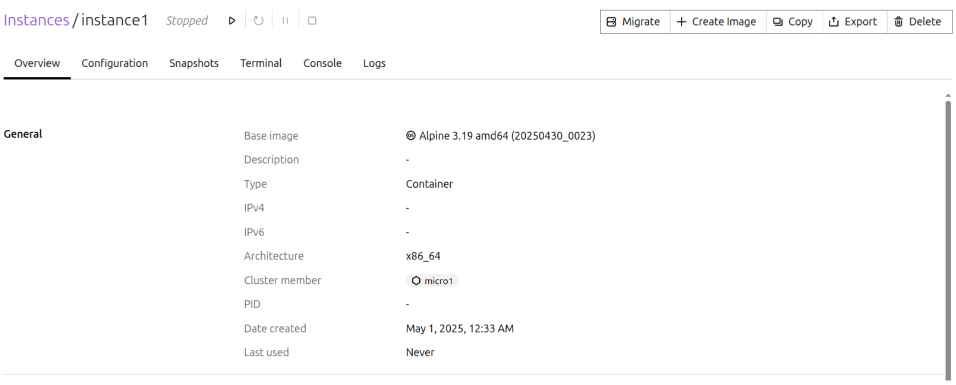
CLI

UI

Use the following command to move the instance to a different pool:

```
lxc move <instance_name> --storage <target_pool_name>
```

Navigate to the overview page of the selected instance, and click on the *Migrate* button in the top right corner.

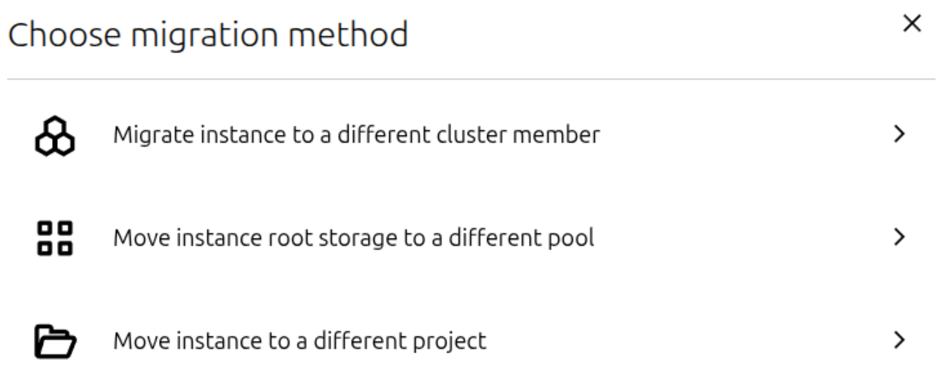


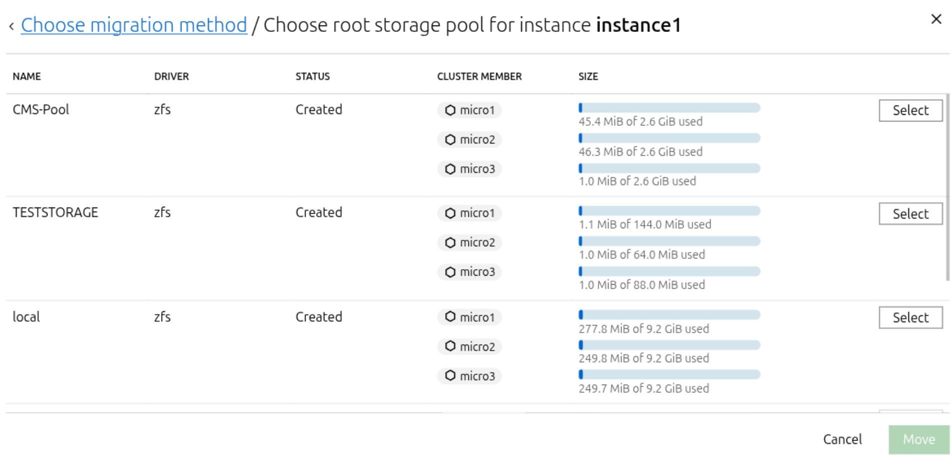
Within the move modal, click *Move instance root storage to a different pool* to view available storage pools to move to.

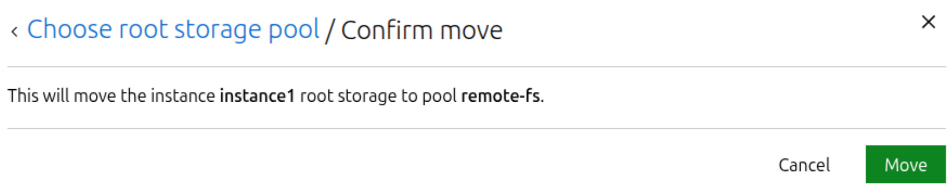
Click *Select* in the row of the target storage pool for the move.

On the resulting confirmation modal, click *Move* to finalize the process.

How to export and move custom storage volumes:


Choose migration method
Migrate instance to a different cluster member
Move instance root storage to a different pool
Move instance to a different project


Choose migration method / Choose root storage pool for instance instance1
NAME
DRIVER
STATUS
CLUSTER MEMBER
SIZE
CMS-Pool
zfs
Created
micro1
45.4 MiB of 2.6 GiB used
micro2
46.3 MiB of 2.6 GiB used
micro3
1.0 MiB of 2.6 GiB used
Select
TESTSTORAGE
zfs
Created
micro1
1.1 MiB of 144.0 MiB used
micro2
1.0 MiB of 64.0 MiB used
micro3
1.0 MiB of 88.0 MiB used
Select
local
zfs
Created
micro1
277.8 MiB of 9.2 GiB used
micro2
249.8 MiB of 9.2 GiB used
micro3
249.7 MiB of 9.2 GiB used
Select
Cancel
Move


Choose root storage pool / Confirm move
This will move the instance instance1 root storage to pool remote-fs.
Cancel
Move

How to back up custom storage volumes

There are different ways of backing up your custom storage volumes:

- *Use snapshots for volume backup* (page 200)
- *Use export files for volume backup* (page 203)
- *Copy custom storage volumes* (page 206)

Which method to choose depends both on your use case and on the storage driver you use.

In general, snapshots are quick and space efficient (depending on the storage driver), but they are stored in the same storage pool as the volume and therefore not too reliable. Export files can be stored on different disks and are therefore more reliable. They can also be used to restore the volume into a different storage pool. If you have a separate, network-connected LXD server available, regularly copying volumes to this other server gives high reliability as well, and this method can also be used to back up snapshots of the volume.

> **Note**
>
> Custom storage volumes might be attached to an instance, but they are not part of the instance. Therefore, the content of a custom storage volume is not stored when you *back up your instance* (page 127). You must back up the data of your storage volume separately.

Use snapshots for volume backup

A snapshot saves the state of the storage volume at a specific time, which makes it easy to restore the volume to a previous state. It is stored in the same storage pool as the volume itself.

Most storage drivers support optimized snapshot creation (see *Feature comparison* (page 571)). For these drivers, creating snapshots is both quick and space-efficient. For the `dir` driver, snapshot functionality is available but not very efficient. For the `lvm` driver, snapshot creation is quick, but restoring snapshots is efficient only when using thin-pool mode.

Create a snapshot of a custom storage volume

CLI

UI

Use the following command to create a snapshot for a custom storage volume:

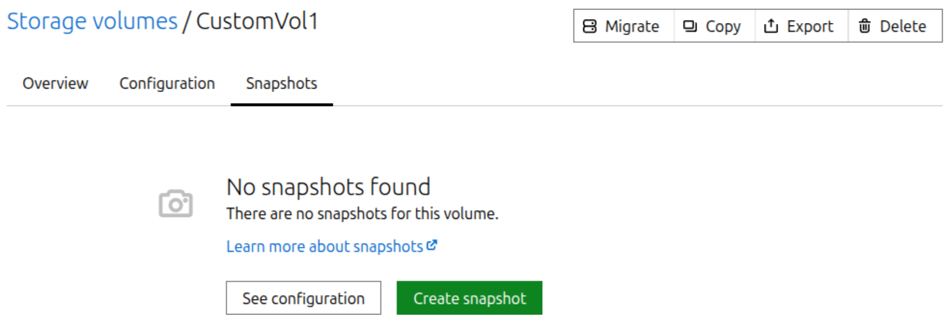
```
lxc storage volume snapshot <pool_name> <volume_name> [<snapshot_name>]
```

The snapshot name is optional. If you don't specify one, the name follows the naming pattern defined in `snapshots.pattern`.

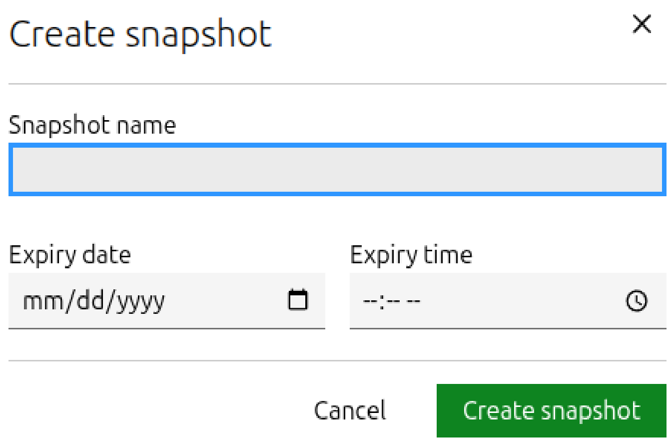
Add the `--reuse` flag in combination with a snapshot name to replace an existing snapshot.

By default, snapshots are kept forever, unless the `snapshots.expiry` configuration option is set. To retain a specific snapshot even if a general expiry time is set, use the `--no-expiry` flag.

To create a snapshot of a custom storage volume, navigate to the *Snapshots* tab for the target volume and click *Create snapshot*.



In the modal that appears, you can provide the snapshot with a name and expiry date and time. If the name is left blank, a name is automatically assigned to the snapshot based on the global snapshot configuration. If the expiry date and time are left blank, the snapshot does not expire.



View, edit or delete snapshots

CLI

UI

Use the following command to display the snapshots for a storage volume:

```
lxc storage volume info <pool_name> <volume_name>
```

You can view or modify snapshots in a similar way to custom storage volumes, by referring to the snapshot with `<volume_name>/<snapshot_name>`.

To show information about a snapshot, use the following command:

```
lxc storage volume show <pool_name> <volume_name>/<snapshot_name>
```

To edit a snapshot (for example, to add a description or change the expiry date), use the following command:

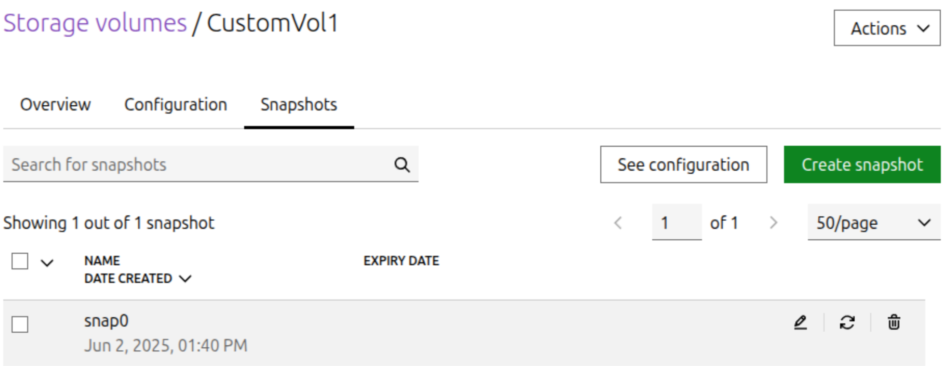
```
lxc storage volume edit <pool_name> <volume_name>/<snapshot_name>
```

To delete a snapshot, use the following command:

```
lxc storage volume delete <pool_name> <volume_name>/<snapshot_name>
```

To view, edit or delete a storage volume snapshot, navigate to the *Snapshots* tab for the target volume.

Hover over a snapshot row to highlight possible actions, including *edit*, *restore* and *delete*.



Schedule snapshots of a custom storage volume

CLI

UI

You can configure a custom storage volume to automatically create snapshots at specific times. To do so, set the `snapshots.schedule` configuration option for the storage volume (see *Configure storage volume settings* (page 190)).

For example, to configure daily snapshots, use the following command:

```
lxc storage volume set <pool_name> <volume_name> snapshots.schedule @daily
```

To configure taking a snapshot every day at 6 am, use the following command:

```
lxc storage volume set <pool_name> <volume_name> snapshots.schedule "0 6 * * *"
```

When scheduling regular snapshots, consider setting an automatic expiry (`snapshots.expiry`) and a naming pattern for snapshots (`snapshots.pattern`). See the *Storage drivers* (page 521) documentation for more information about those configuration options.

To schedule a snapshot of a storage volume, navigate to the *Overview* tab of the target volume. Select the *Snapshots* tab and click *See configuration*.

In the resulting modal, you can override the default schedule for automatic volume snapshots. Select the time frame via the Cron expression syntax[116] or a time interval.

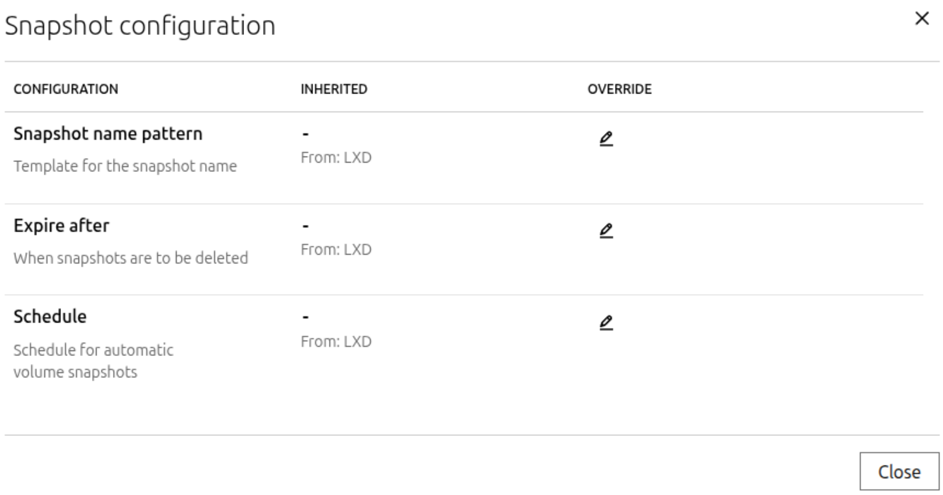
Restore a snapshot of a custom storage volume

CLI

UI

You can restore a custom storage volume to the state of any of its snapshots.

[116] https://en.wikipedia.org/wiki/Cron#Cron_expression



To do so, you must first stop all instances that use the storage volume. Then use the following command:

```
lxc storage volume restore <pool_name> <volume_name> <snapshot_name>
```

You can also restore a snapshot into a new custom storage volume, either in the same storage pool or in a different one (even a remote storage pool). To do so, use the following command:

```
lxc storage volume copy <source_pool_name>/<source_volume_name>/<source_snapshot_
name> <target_pool_name>/<target_volume_name>
```

To restore a storage volume, navigate to the *Snapshots* tab for the target volume, then hover over a snapshot row to view possible actions. Click the *restore* button.

Use export files for volume backup

You can export the full content of your custom storage volume to a standalone file that can be stored at any location. For highest reliability, store the backup file on a different file system to ensure that it does not get lost or corrupted.

Export a custom storage volume

CLI

UI

Use the following command to export a custom storage volume to a compressed file (for example, `/path/to/my-backup.tgz`):

```
lxc storage volume export <pool_name> <volume_name> [<file_path>]
```

If you do not specify a file path, the export file is saved as `backup.tar.gz` in the working directory.

> **Warning**
>
> If the output file already exists, the command overwrites the existing file without warning.

You can add any of the following flags to the command:

`--compression`
: By default, the output file uses `gzip` compression. You can specify a different compression algorithm (for example, `bzip2`) or turn off compression with `--compression=none`.

`--optimized-storage`
: If your storage pool uses the `btrfs` or the `zfs` driver, add the `--optimized-storage` flag to store the data as a driver-specific binary blob instead of an archive of individual files. In this case, the export file can only be used with pools that use the same storage driver.

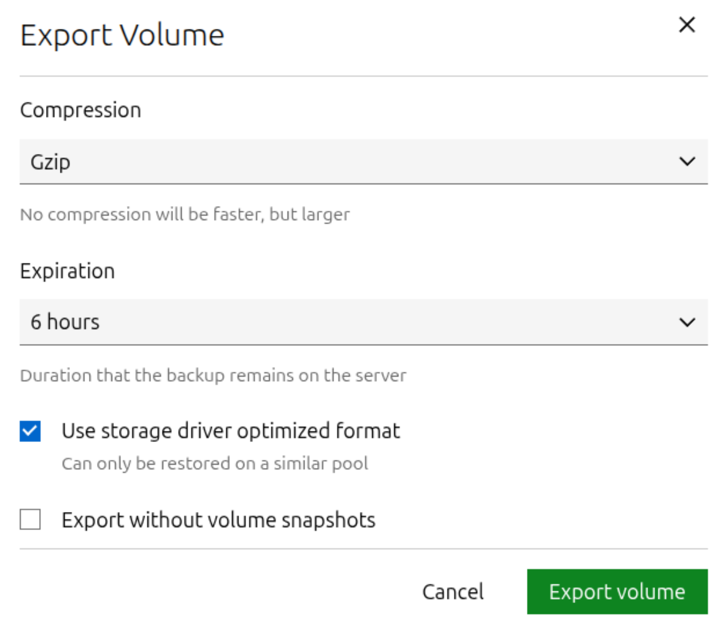
  Exporting a volume in optimized mode is usually quicker than exporting the individual files. Snapshots are exported as differences from the main volume, which decreases their size (quota) and makes them easily accessible.

`--export-version`
: If you intend to import the backup to an older version of LXD, set the version to `1` which will use the original (old) backup metadata format. Backups using the old format can always be imported on newer versions of LXD. If the flag is not specified and the server has support for the `backup_metadata_version` API extension, version `2` is used by default.

`--volume-only`
: By default, the export file contains all snapshots of the storage volume. Add this flag to export the volume without its snapshots.

To export a storage volume, navigate to the *Overview* tab for the target volume and select the *Export* button.

In the resulting modal, configure the export file for the storage volume, including its compression mode and expiration.



Restore a custom storage volume from an export file

CLI

UI

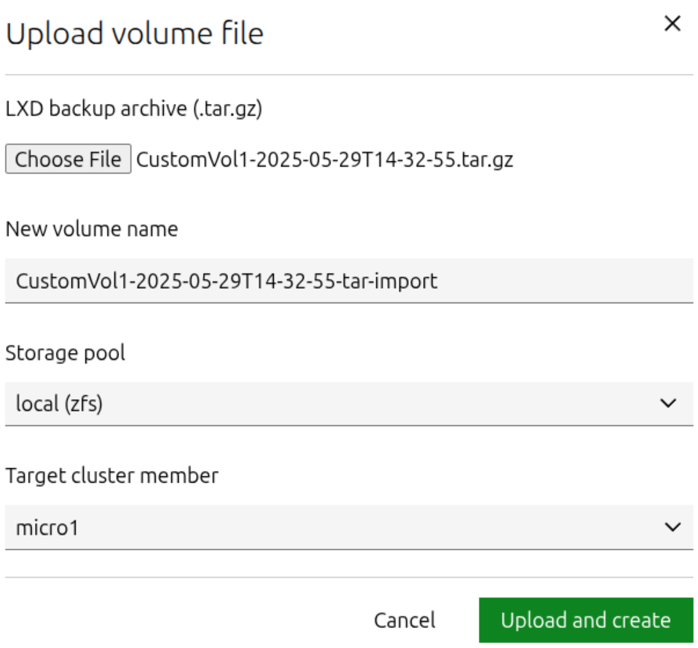
You can import an export file (for example, `/path/to/my-backup.tgz`) as a new custom storage volume. To do so, use the following command:

```
lxc storage volume import <pool_name> <file_path> [<volume_name>]
```

If you do not specify a volume name, the original name of the exported storage volume is used for the new volume. If a volume with that name already (or still) exists in the specified storage pool, the command returns an error. In that case, either delete the existing volume before importing the backup or specify a different volume name for the import.

To restore a storage volume from an export file, select *Volumes* from the main navigation, then click the *Create volume* button.

Choose the volume file to upload, and select the storage pool for the volume to be created using the export file.



For clustered environments only

Within a clustered environment, if a cluster-member-specific storage pool is selected, you can also configure a target cluster member for the volume.

How to move or copy storage volumes

You can *copy* (page 206) or *move* (page 206) custom storage volumes from one storage pool to another, or copy or rename them within the same storage pool.

To move instance storage volumes from one storage pool to another, *move the corresponding instance* (page 208) to another pool.

When copying or moving a volume between storage pools that use different drivers, the volume is automatically converted.

Copy custom storage volumes

CLI

UI

Use the following command to copy a custom storage volume:

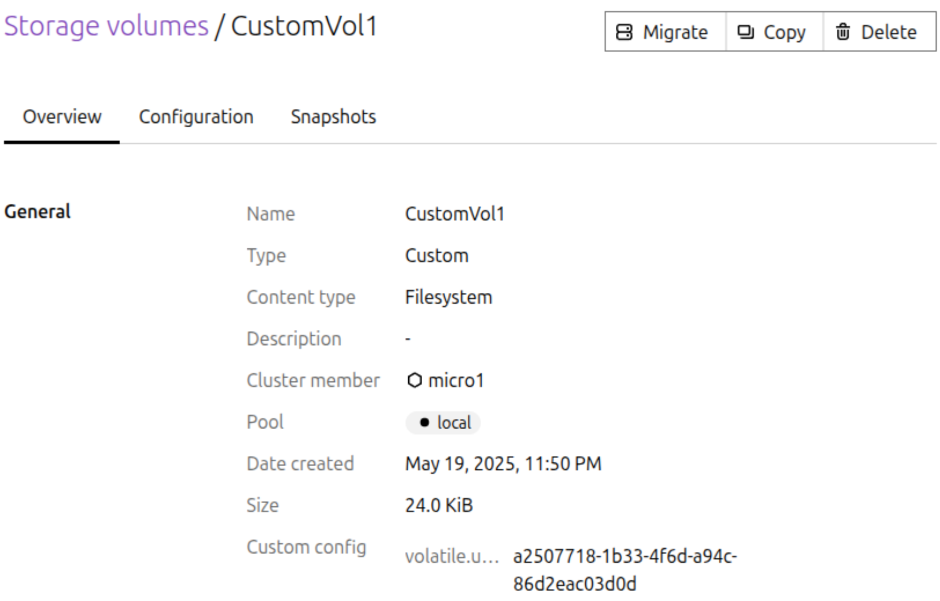
```
lxc storage volume copy <source_pool_name>/<source_volume_name> <target_pool_name>
/<target_volume_name>
```

Add the `--volume-only` flag to copy only the volume and skip any snapshots that the volume might have. If the volume already exists in the target location, use the `--refresh` flag to update the copy (see *Optimized volume transfer* (page 572) for the benefits).

Specify the same pool as the source and target pool to copy the volume within the same storage pool. You must specify different volume names for source and target in this case.

When copying from one storage pool to another, you can either use the same name for both volumes or rename the new volume.

To copy a custom storage volume, navigate to the *Overview* page of the storage volume you wish to copy, and click *Copy*.



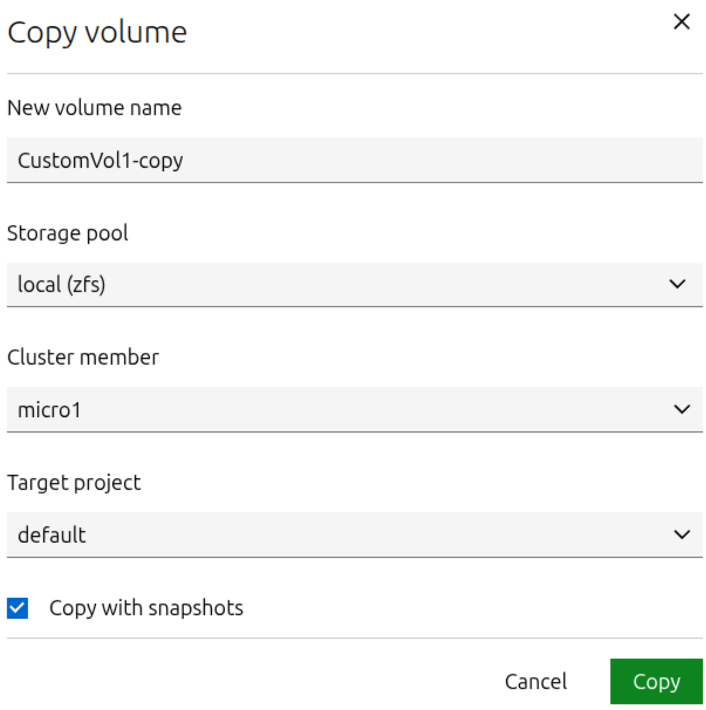
In the *Copy volume* modal, you can define a new name for the copied volume as well as a number of other attributes.

Move or rename custom storage volumes

CLI

UI

Before you can move or rename a custom storage volume, all instances that use it must be *stopped* (page 94).



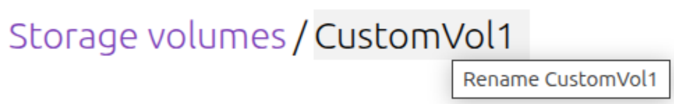
Use the following command to move or rename a storage volume:

```
lxc storage volume move <source_pool_name>/<source_volume_name> <target_pool_name>
/<target_volume_name>
```

Specify the same pool as the source and target pool to rename the volume while keeping it in the same storage pool. You must specify different volume names for source and target in this case.

When moving from one storage pool to another, you can either use the same name for both volumes or rename the new volume.

To rename a custom storage volume, navigate to its *Overview* page and select its name in the header to edit it.



Copy or migrate between cluster members

CLI

UI

For most storage drivers (except for `ceph` and `ceph-fs`), storage volumes exist only on the cluster member for which they were created.

To copy or migrate a custom storage volume from one cluster member to another, add the `--target` and `--destination-target` flags to specify the source cluster member and the target cluster member, respectively.

You can use the LXD UI to copy storage volumes between cluster members, but not to migrate them.

To copy a storage volume, navigate to the *Overview* page of the storage volume within a clustered environment, then click *Copy*.

In the *Copy volume* modal, select the target cluster member from the *Cluster member* dropdown.

Copy or move between projects

CLI

UI

Add the `--target-project` to copy or move a custom storage volume to a different project.

To copy a storage volume between projects, navigate to the *Overview* page of the storage volume, then click *Copy*.

In the *Copy volume* modal, select the target from the *Target project* dropdown.

Copy or migrate between LXD servers

You can copy a custom volume from one LXD server to another, or migrate it (move it between servers), by specifying the remote for each pool:

```
lxc storage volume copy <source_remote>:<source_pool_name>/<source_volume_name>
<target_remote>:<target_pool_name>/<target_volume_name>
lxc storage volume move <source_remote>:<source_pool_name>/<source_volume_name>
<target_remote>:<target_pool_name>/<target_volume_name>
```

You can add the `--mode` flag to choose a transfer mode, depending on your network setup:

`pull` (default)
: Instruct the target server to pull the respective storage volume.

`push`
: Push the storage volume from the source server to the target server.

`relay`
: Pull the storage volume from the source server to the local client, and then push it to the target server.

If the volume already exists in the target location, use the `--refresh` flag to update the copy (see *Optimized volume transfer* (page 572) for the benefits).

Move instance storage volumes to another pool

To move an instance storage volume to another storage pool, *stop the instance* (page 94) that contains the storage volume you want to move.

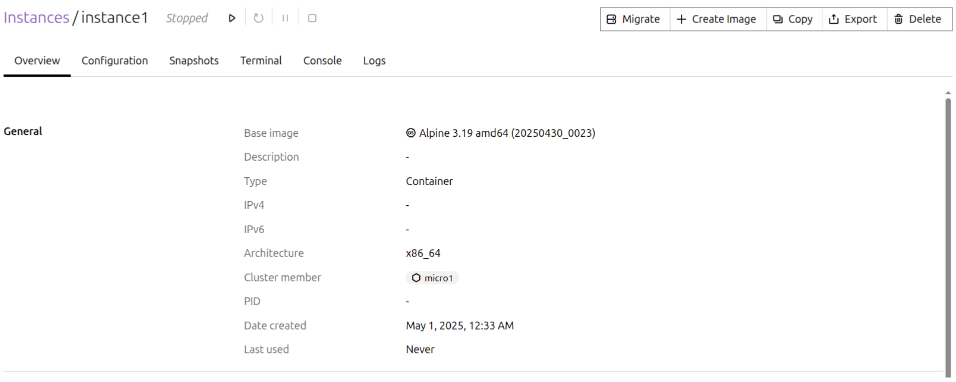
CLI

UI

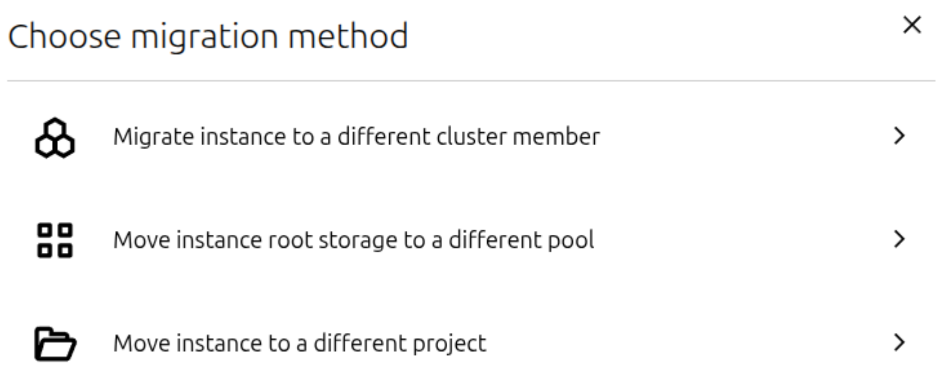
Use the following command to move the instance to a different pool:

```
lxc move <instance_name> --storage <target_pool_name>
```

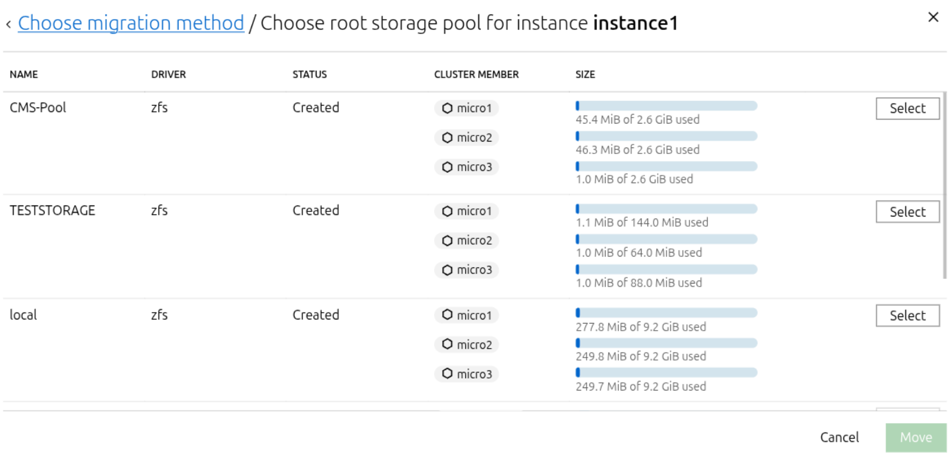
Navigate to the overview page of the selected instance, and click on the *Migrate* button in the top right corner.



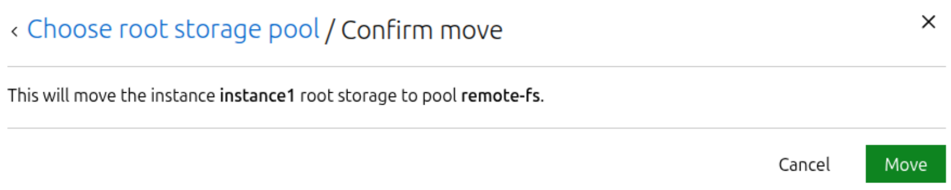
Within the move modal, click *Move instance root storage to a different pool* to view available storage pools to move to.



Click *Select* in the row of the target storage pool for the move.



On the resulting confirmation modal, click *Move* to finalize the process.



Related topics

Explanation:

- *Storage pools, volumes, and buckets* (page 349)

Reference:

- *Storage drivers* (page 521)

2.2.5. Networking

The following how-to guides cover common operations related to networking.

How to create and configure a network:

How to create a network

To create a managed network, use the *lxc network* (page 790) command and its subcommands. Append `--help` to any command to see more information about its usage and available flags.

Network types

The following network types are available:

Network type	Documentation	Configuration options
`bridge`	*Bridge network* (page 573)	*Configuration options* (page 574)
`ovn`	*OVN network* (page 587)	*Configuration options* (page 588)
`macvlan`	*Macvlan network* (page 593)	*Configuration options* (page 594)
`sriov`	*SR-IOV network* (page 600)	*Configuration options* (page 600)
`physical`	*Physical network* (page 595)	*Configuration options* (page 595)

Create a network

CLI

UI

Use the following command to create a network:

```
lxc network create <name> --type=<network_type> [configuration_options...]
```

See *Network types* (page 210) for a list of available network types and links to their configuration options.

If you do not specify a `--type` argument, the default type of `bridge` is used.

Create a network in a cluster

If you are running a LXD cluster and want to create a network, you must create the network for each cluster member separately. The reason for this is that the network configuration, for example, the name of the parent network interface, might be different between cluster members.

Therefore, you must first create a pending network on each member with the `--target=<cluster_member>` flag and the appropriate configuration for the member. Make sure to use the same network name for all members. Then create the network without specifying the `--target` flag to actually set it up.

For example, the following series of commands sets up a physical network with the name `UPLINK` on three cluster members:

```
~$ lxc network create UPLINK --type=physical parent=br0 --target=vm01 Network
UPLINK pending on member vm01 ~$ lxc network create UPLINK --type=physical
parent=br0 --target=vm02 Network UPLINK pending on member vm02 ~$ lxc network
create UPLINK --type=physical parent=br0 --target=vm03 Network UPLINK pending
on member vm03 ~$ lxc network create UPLINK --type=physical Network UPLINK
created
```

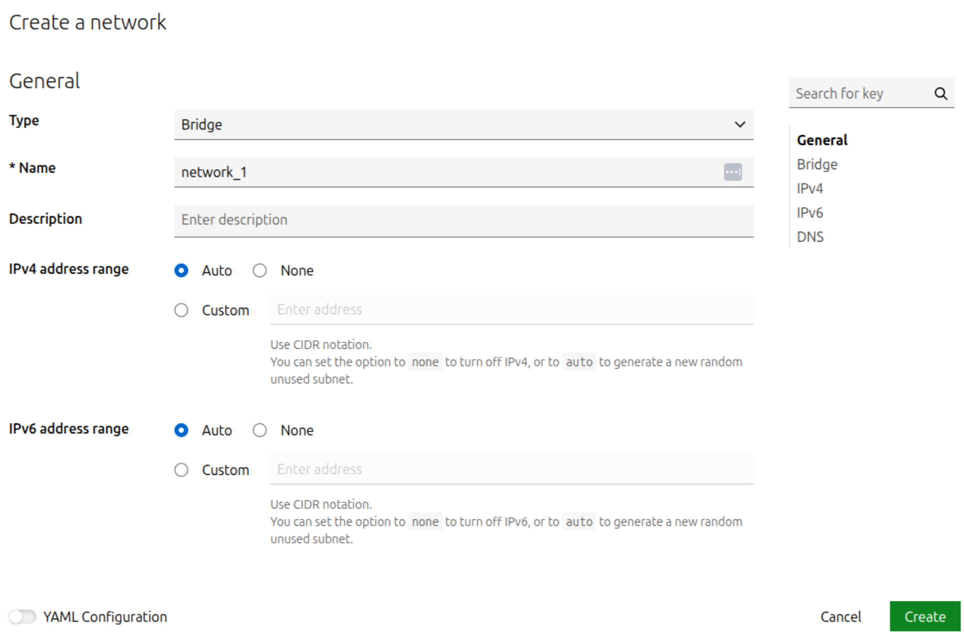
Also see *How to configure networks for a cluster* (page 288).

From the main navigation, select *Networks*.

On the resulting page, click *Create network* in the upper-right corner.

You can then configure the network name and type, as well as other attributes. Optional additional attributes are split into the categories *Bridge*, *IPv4*, *IPv6* and *DNS*, which can be seen in the submenu on the right.

Click *Create* to create the network.



Attach a network to an instance

CLI

UI

After creating a managed network, you can attach it to an instance as a *NIC device* (page 449).

To do so, use the following command:

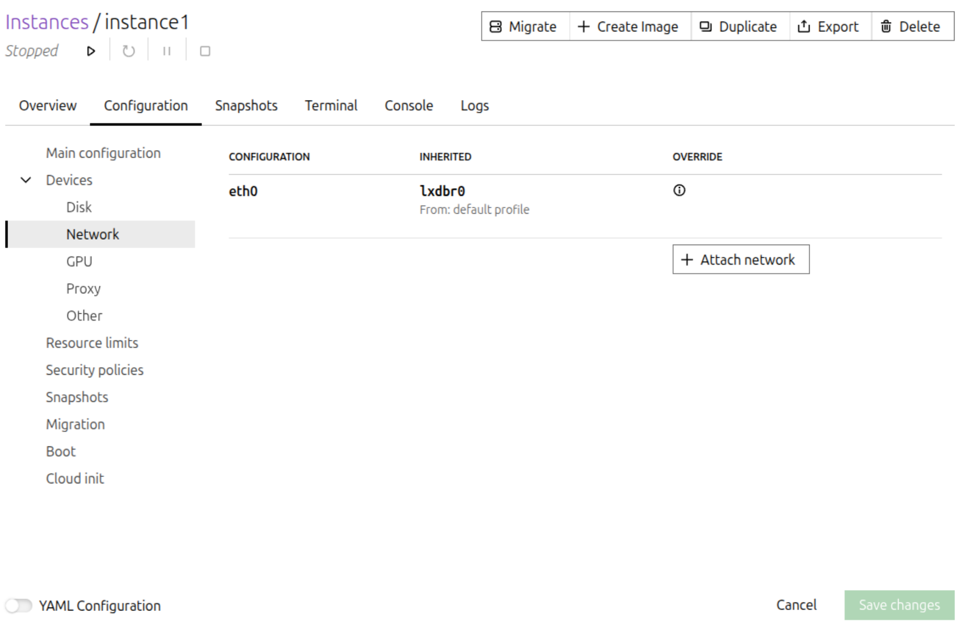
```
lxc network attach <network_name> <instance_name> [<device_name>] [<interface_
name>]
```

The device name and the interface name are optional, but we recommend specifying at least the device name. If not specified, LXD uses the network name as the device name, which might be confusing and cause problems. For example, LXD images perform IP auto-configuration on the `eth0` interface, which does not work if the interface is called differently.

For example, to attach the network `my-network` to the instance `my-instance` as `eth0` device, enter the following command:

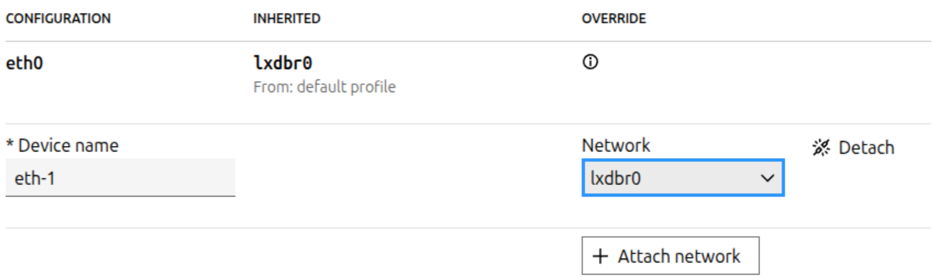
```
lxc network attach my-network my-instance eth0
```

When *creating* (page 73) or *configuring an instance* (page 84), go to the *Devices* section in the left-hand submenu, then select *Network* to view and edit the networks linked to the instance.



Click the *Attach network* button to add a new network. From here, you can select an existing network from the *Network* dropdown and assign it a device name.

If configuring an instance, select *Save changes* to save your changes. If creating an instance, select *Create* to create your instance.



Attach the network as a device

The *lxc network attach* (page 799) command is a shortcut for adding a NIC device to an instance. Alternatively, you can add a NIC device based on the network configuration in the usual way:

```
lxc config device add <instance_name> <device_name> nic network=<network_name>
```

When using this way, you can add further configuration to the command to override the default settings for the network if needed. See *NIC device* (page 449) for all available device options.

How to configure a network

CLI

UI

To configure an existing network, use either the *lxc network set* (page 830) and *lxc network unset* (page 831) commands (to configure single settings) or the `lxc network edit` command (to edit the full configuration). To configure settings for specific cluster members, add the `--target` flag.

For example, the following command configures a DNS server for a physical network:

```
lxc network set UPLINK dns.nameservers=8.8.8.8
```

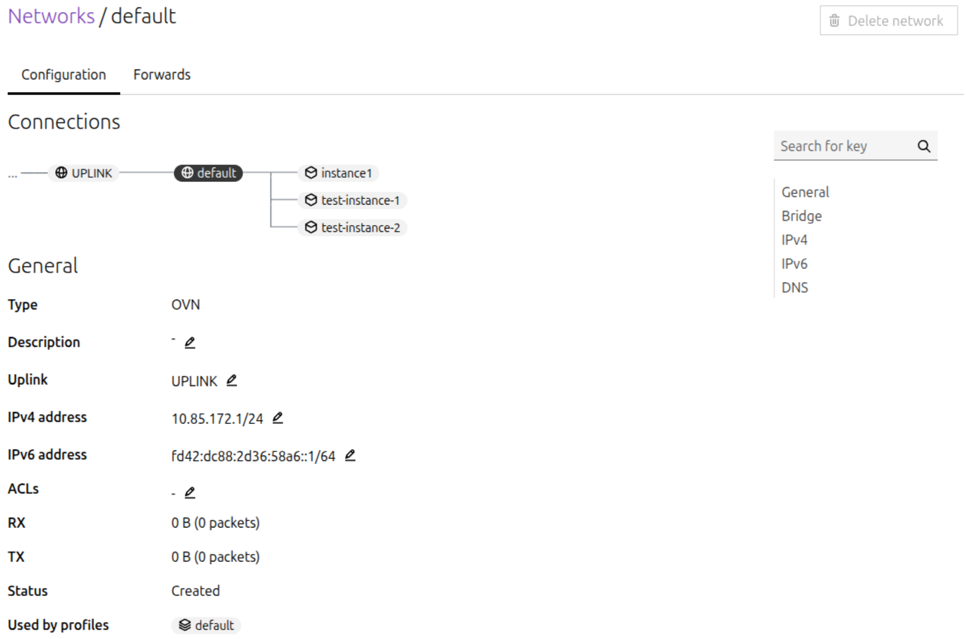
The available configuration options differ depending on the network type. See *Network types* (page 210) for links to the configuration options for each network type.

To edit the configuration of a network, navigate to the overview page for the network, and observe its attributes and settings.

Within the Configuration tab, you can edit key settings of the network by clicking on the *Edit* pencil icon inline with the desired configuration setting.

There are separate commands to configure advanced networking features. See the following documentation:

- *How to configure network ACLs* (page 216)
- *How to configure network forwards* (page 235)
- *How to configure network load balancers* (page 271)
- *How to configure network zones* (page 252)
- *How to create OVN peer routing relationships* (page 276) (OVN only)



How to configure specific networking features:

How to configure LXD as a BGP server

> **Note**
>
> The BGP server feature is available for the *Bridge network* (page 573) and the *Physical network* (page 595). These network types are often used as the uplink network for an *OVN network* (page 587), and you must configure the BGP peers on the uplink network. See *Configure BGP peers for OVN networks* (page 215) for instructions.

BGP (Border Gateway Protocol) is a protocol that allows exchanging routing information between autonomous systems.

If you want to directly route external addresses to specific LXD servers or instances, you can configure LXD as a BGP server. LXD will then act as a BGP peer and advertise relevant routes and next hops to external routers, for example, your network router. It automatically establishes sessions with upstream BGP routers and announces the addresses and subnets that it's using.

The BGP server feature can be used to allow a LXD server or cluster to directly use internal/external address space by getting the specific subnets or addresses routed to the correct host. This way, traffic can be forwarded to the target instance.

For bridge networks, the following addresses and networks are being advertised:

- Network `ipv4.address` or `ipv6.address` subnets (if the matching `nat` property isn't set to `true`)
- Network `ipv4.nat.address` or `ipv6.nat.address` subnets (if the matching `nat` property is set to `true`)
- Network forward addresses

- Addresses or subnets specified in `ipv4.routes.external` or `ipv6.routes.external` on an instance NIC that is connected to the bridge network

Make sure to add your subnets to the respective configuration options. Otherwise, they won't be advertised.

For physical networks, no addresses are advertised directly at the level of the physical network. Instead, the networks, forwards and routes of all downstream networks (the networks that specify the physical network as their uplink network through the `network` option) are advertised in the same way as for bridge networks.

> **Note**
>
> At this time, it is not possible to announce only some specific routes/addresses to particular peers. If you need this, filter prefixes on the upstream routers.

Configure the BGP server

To configure LXD as a BGP server, set the following server configuration options on all cluster members:

- *core.bgp_address* (page 401) - the IP address for the BGP server
- *core.bgp_asn* (page 401) - the ASN (Autonomous System Number) for the local server
- *core.bgp_routerid* (page 401) - the unique identifier for the BGP server

For example, set the following values:

```
lxc config set core.bgp_address=192.0.2.50:179
lxc config set core.bgp_asn=65536
lxc config set core.bgp_routerid=192.0.2.50
```

Once these configuration options are set, LXD starts listening for BGP sessions.

Configure next-hop (`bridge` only)

For bridge networks, you can override the next-hop configuration. By default, the next-hop is set to the address used for the BGP session.

To configure a different address, set `bgp.ipv4.nexthop` or `bgp.ipv6.nexthop`.

Configure BGP peers for OVN networks

If you run an OVN network with an uplink network (`physical` or `bridge`), the uplink network is the one that holds the list of allowed subnets and the BGP configuration. Therefore, you must configure BGP peers on the uplink network that contain the information that is required to connect to the BGP server.

Set the following configuration options on the uplink network:

- `bgp.peers.<name>.address` - the peer address to be used by the downstream networks
- `bgp.peers.<name>.asn` - the ASN for the local server
- `bgp.peers.<name>.password` - an optional password for the peer session

- `bgp.peers.<name>.holdtime` - an optional hold time for the peer session (in seconds)

Once the uplink network is configured, downstream OVN networks will get their external subnets and addresses announced over BGP. The next-hop is set to the address of the OVN router on the uplink network.

How to configure network ACLs

> **Note**
>
> Network ACLs are available for the *OVN NIC type* (page 462), the *OVN network* (page 587) and the *Bridge network* (page 573) (with some exceptions; see *Bridge limitations* (page 234)).

Network ACLs (Access Control Lists) define rules for controlling traffic:

- Between instances connected to the same network
- To and from other networks

Network ACLs can be assigned directly to the NIC (Network Interface Controller) of an instance, or to a network. When assigned to a network, the ACL applies indirectly to all NICs connected to that network.

When an ACL is assigned to multiple instance NICs, either directly or indirectly, those NICs form a logical port group. You can use the name of that ACL to refer to that group in the traffic rules of other ACLs. For more information, see: *Subject name selectors (ACL groups)* (page 224).

List ACLs

CLI

API

To list all ACLs, run:

```
lxc network acl list
```

To list all ACLs, query the `GET /1.0/network-acls` endpoint:

```
lxc query --request GET /1.0/network-acls
```

You can also use *recursion* (page 621) to list the ACLs with a higher level of detail:

```
lxc query --request GET /1.0/network-acls?recursion=1
```

Show an ACL

CLI

API

To show details about a specific ACL, run:

```
lxc network acl show <ACL-name>
```

Example:

```
lxc network acl show my-acl
```

For details about a specific ACL, query the `GET /1.0/network-acls/{ACL-name}` endpoint`:

```
lxc query --request GET /1.0/network-acls/{ACL-name}
```

Example:

```
lxc query --request GET /1.0/network-acls/my-acl
```

Create an ACL

Name requirements

Network ACL names must meet the following requirements:

- Must be between 1 and 63 characters long.
- Can contain only ASCII letters (a–z, A–Z), numbers (0–9), and dashes (-).
- Cannot begin with a digit or a dash.
- Cannot end with a dash.

Instructions

CLI

API

To create an ACL, run:

```
lxc network acl create <ACL-name> [user.KEY=value ...]
```

- You must provide an ACL name that meets the *Name requirements* (page 217).
- You can optionally provide one or more custom `user` keys to store metadata or other information.

ACLs have no rules upon creation via command line, so as a next step, *add rules* (page 220) to the ACL. You can also *edit the ACL configuration* (page 226), or *assign the ACL to a network or NIC* (page 230).

Another way to create ACLs from the command line is to provide a YAML configuration file:

```
lxc network acl create <ACL-name> < <filename.yaml>
```

This file can include any other *ACL properties* (page 219), including the `egress` and `ingress` properties for defining *ACL rules* (page 220). See the second example in the set below.

Examples

Create an ACL with the name `my-acl` and an optional custom user key:

```
lxc network acl create my-acl user.my-key=my-value
```

Create an ACL using a YAML configuration file:

First, create a file named `config.yaml` with the following content:

```
description: Allow web traffic from internal network
config:
  user.owner: devops
ingress:
  - action: allow
    description: Allow HTTP/HTTPS from internal
    protocol: tcp
    source: "@internal"
    destination_port: "80,443"
    state: enabled
```

Note that the custom user keys are stored under the `config` property.

The following command creates an ACL from that file's configuration:

```
lxc network acl create my-acl < config.yaml
```

To create an ACL, query the `POST /1.0/network-acls` endpoint:

```
lxc query --request POST /1.0/network-acls --data '{
  "name": "<ACL-name>",
  "config": {
    "user.<custom-key-name>": "<custom-key-value>"
  },
  "description": "<description of the ACL>",
  "egress": [{<egress rule object>}, {<another egress rule object>, ...}],
  "ingress": [{<ingress rule object>}, {<another ingress rule object>, ...}]
}'
```

- You must provide an ACL name that meets the *Name requirements* (page 217).
- You can optionally provide one or more custom `config.user.*` keys to store metadata or other information.
- The `ingress` and `egress` lists contain rules for inbound and outbound traffic. See *ACL rules* (page 220) for details.

Examples

Create an ACL with the name `my-acl`, a custom user key of `my-key`, and a `description`:

```
lxc query --request POST /1.0/network-acls --data '{
  "name": "my-acl",
  "config": {
    "user.my-key": "my-value"
```

(continues on next page)

```
  },
  "description": "Web servers"
}'
```

Create an ACL with the name `my-acl` and an `ingress` rule:

```
lxc query --request POST /1.0/network-acls --data '{
  "name": "my-acl",
  "ingress": [
    {
      "action": "drop",
      "state": "enabled"
    }
  ]
}'
```

ACL properties

ACLs have the following properties: `config` User-provided free-form key/value pairs (page 219)

Key:	`config`
Type:	string set
Required:	no

The only supported keys are `user.*` custom keys.

`description` Description of the network ACL (page 219)

Key:	`description`
Type:	string
Required:	no

`egress` Egress traffic rules (page 219)

Key:	`egress`
Type:	rule list
Required:	no

`ingress` Ingress traffic rules (page 219)

Key:	`ingress`
Type:	rule list
Required:	no

`name` Unique name of the network ACL in the project (page 219)

Key:	name
Type:	string
Required:	yes

ACL rules

Each ACL contains two lists of rules:

- Rules in the `egress` list apply to outbound traffic from the NIC.
- Rules in the `ingress` list apply to inbound traffic to the NIC.

For both `egress` and `ingress`, the rule configuration looks like this:

YAML

JSON

```yaml
action: <allow|reject|drop>
description: <description>
destination: <destination-IP-range>
destination_port: <destination-port-number>
icmp_code: <ICMP-code>
icmp_type: <ICMP-type>
protocol: <icmp4|icmp6|tcp|udp>
source: <source-of-traffic>
source_port: <source-port-number>
state: <enabled|disabled|logged>
```

```json
{
  "action": "<allow|reject|drop>",
  "description": "<description>",
  "destination": "<destination-IP-range>",
  "destination_port": "<destination-port-number>",
  "icmp_code": "<ICMP-code>",
  "icmp_type": "<ICMP-type>",
  "protocol": "<icmp4|icmp6|tcp|udp>",
  "source": "<source-of-traffic>",
  "source_port": "<source-port-number>",
  "state": "<enabled|disabled|logged>"
}
```

- The `action` property is required.
- The `state` property defaults to `"enabled"` if unset.
- The `source` and `destination` properties can be specified as one or more CIDR blocks, IP ranges, or *selectors* (page 224). If left empty, they match any source or destination. Comma-separate multiple values.
- If the `protocol` is unset, it matches any protocol.
- The `"destination_port"` and `"source_port"` properties and `"icmp_code"` and `"icmp_type"` properties are mutually exclusive sets. Although both sets are shown

in the same rule above to demonstrate the syntax, they never appear together in practice.

- The "destination_port" and "source_port" properties are only available when the "protocol" for the rule is "tcp" or "udp".
- The "icmp_code"[117] and "icmp_type"[118] properties are only available when the "protocol" is "icmp4" or "icmp6".

- The "state" is "enabled" by default. The "logged" value is used to *log traffic* (page 225) to a rule.

For more information, see: *Rule properties* (page 222).

Add a rule

CLI

API

To add a rule to an ACL, run:

```
lxc network acl rule add <ACL-name> <egress|ingress> [properties...]
```

Example

Add an egress rule with an action of drop to my-acl:

```
lxc network acl rule add my-acl egress action=drop
```

There is no specific endpoint for adding a rule. Instead, you must *edit the full ACL* (page 226), which contains the egress and ingress lists.

Remove a rule

CLI

API

To remove a rule from an ACL, run:

```
lxc network acl rule remove <ACL-name> <egress|ingress> [properties...]
```

You must either specify all properties needed to uniquely identify a rule or add --force to the command to delete all matching rules.

There is no specific endpoint for removing a rule. Instead, you must *edit the full ACL* (page 226), which contains the egress and ingress lists.

Edit a rule

You cannot edit a rule directly. Instead, you must *edit the full ACL* (page 226), which contains the egress and ingress lists.

[117] https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml#icmp-parameters-codes
[118] https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml#icmp-parameters-types

Rule ordering and application of actions

ACL rules are defined as lists, but their order within the list does not affect how they are applied.

LXD automatically prioritizes rules based on the action property, in the following order:

- `drop`
- `reject`
- `allow`
- The default action for unmatched traffic (defaults to `reject`, see *Configure default actions* (page 232))

When you assign multiple ACLs to a NIC, you do not need to coordinate rule order across them. As soon as a rule matches, its action is applied and no further rules are evaluated.

Rule properties

ACL rules have the following properties: `action` Action to take for matching traffic (page 222)

Key:	`action`
Type:	string
Required:	yes

Possible values are `allow`, `reject`, and `drop`.

`description` Description of the rule (page 222)

Key:	`description`
Type:	string
Required:	no

`destination` Comma-separated list of destinations (page 222)

Key:	`destination`
Type:	string
Required:	no

Destinations can be specified as CIDR or IP ranges, destination subject name selectors (for egress rules), or be left empty for any.

`destination_port` Destination ports or port ranges (page 222)

Key:	`destination_port`
Type:	string
Required:	no

This option is valid only if the protocol is udp or tcp. Specify a comma-separated list of ports or port ranges (start-end inclusive), or leave the value empty for any.

icmp_code ICMP message code (page 223)

Key:	icmp_code
Type:	string
Required:	no

This option is valid only if the protocol is icmp4 or icmp6. Specify the ICMP code number, or leave the value empty for any.

icmp_type Type of ICMP message (page 223)

Key:	icmp_type
Type:	string
Required:	no

This option is valid only if the protocol is icmp4 or icmp6. Specify the ICMP type number, or leave the value empty for any.

protocol Protocol to match (page 223)

Key:	protocol
Type:	string
Required:	no

Possible values are icmp4, icmp6, tcp, and udp. Leave the value empty to match any protocol.

source Comma-separated list of sources (page 223)

Key:	source
Type:	string
Required:	no

Sources can be specified as CIDR or IP ranges, source subject name selectors (for ingress rules), or be left empty for any.

source_port Source ports or port ranges (page 223)

Key:	source_port
Type:	string
Required:	no

This option is valid only if the protocol is udp or tcp. Specify a comma-separated list of ports or port ranges (start-end inclusive), or leave the value empty for any.

state State of the rule (page 223)

Key:	state
Type:	string
Default:	enabled
Required:	yes

Possible values are `enabled`, `disabled`, and `logged`.

Use selectors in rules

> **Note**
>
> This feature is supported only for the *OVN NIC type* (page 462) and the *OVN network* (page 587).

In ACL rules, the `source` and `destination` properties support using selectors instead of CIDR blocks or IP ranges. You can only use selectors in the `source` of `ingress` rules, and in the `destination` of `egress` rules.

Using selectors allows you to define rules for groups of instances instead of managing lists of IP addresses or subnets manually.

There are two types of selectors:

- subject name selectors (ACL groups)
- network subject selectors

Subject name selectors (ACL groups)

When an ACL is assigned to multiple instance NICs, either directly or through their networks, those NICs form a logical port group. You can use the name of that ACL as a *subject name selector* to refer to that group in the egress and ingress lists of other ACLs.

For example, if you have an ACL with the name `my-acl`, you can specify the group of instance NICs that are assigned this ACL as an egress or ingress rule's source by setting `source` to `my-acl`.

Network subject selectors

Use *network subject selectors* to define rules based on the network that the traffic is coming from or going to.

All network subject selectors begin with the `@` symbol. There are two special network subject selectors called `@internal` and `@external`. They represent the network's local and external traffic, respectively.

Here's an example ACL rule (in YAML) that allows all internal traffic with the specified destination port:

```yaml
ingress:
  - action: allow
    description: Allow HTTP/HTTPS from internal
```

(continues on next page)

(continued from previous page)

```
    protocol: tcp
    source: "@internal"
    destination_port: "80,443"
    state: enabled
```

If your network supports *network peers* (page 276), you can reference traffic to or from the peer connection by using a network subject selector in the format `@<network-name>/<peer-name>`. Example:

```
source: "@my-network/my-peer"
```

When using a network subject selector, the network that has the ACL assigned to it must have the specified peer connection.

Log traffic

ACL rules are primarily used to control network traffic between instances and networks. However, they can also be used to log specific types of traffic, which is useful for monitoring or testing rules before enabling them.

To configure a rule so that it only logs traffic, configure its `state` to `logged` when you *add the rule* (page 220) or *edit the ACL* (page 226).

View logs

CLI

API

To display the logs for all `logged` rules in an ACL, run:

```
lxc network acl show-log <ACL-name>
```

To display the logs for all `logged` rules in an ACL, query the `GET /1.0/network-acls/{ACL-name}/log` endpoint:

```
lxc query --request GET /1.0/network-acls/{ACL-name}/log
```

Example

```
lxc query --request GET /1.0/network-acls/my-acl/log
```

Note

If your attempt to view logs returns no data, that means either:

- No `logged` rules have matched any traffic yet.
- The ACL does not contain any rules with a `state` of `logged`.

When displaying logs for an ACL, LXD intentionally displays all existing logs for that ACL, including logs from formerly `logged` rules that are no longer set to log traffic. Thus, if you see logs from an ACL rule, that does not necessarily mean that its `state` is *currently* set to `logged`.

Edit an ACL

Rename an ACL

Requirements:

- You can only rename an ACL that is not currently *assigned to a NIC or network* (page 230).
- The new name must meet the *Name requirements* (page 217).

CLI

API

To rename an ACL, run:

```
lxc network acl rename <old-ACL-name> <new-ACL-name>
```

To rename an ACL, query the `POST /1.0/network-acls/{ACL-name}` endpoint:

```
lxc query --request POST /1.0/network-acls/{ACL-name} --data '{
  "name": "<new-ACL-name>"
}'
```

Example

Rename an ACL named `web-traffic` to `internal-web-traffic`:

```
lxc query --request POST /1.0/network-acls/web-traffic --data '{
  "name": "internal-web-traffic"
}'
```

Edit other properties

CLI

API

Run:

```
lxc network acl edit <ACL-name>
```

This command opens the ACL configuration in YAML format for editing. You can edit any part of the configuration *except* for the ACL name, including the custom user keys.

You can update any ACL property except for `name`, including the custom user keys, by querying the `PUT /1.0/network-acls/{ACL-name}` endpoint:

```
lxc query --request PUT /1.0/network-acls/{ACL-name} --data '{
  "config": {
    "user.<custom key name>": "<custom key value>"
```

(continues on next page)

```
  },
  "description": "<description of the ACL>",
  "egress": [<egress rule>, <another egress rule...>,...],
  "ingress": [<ingress rule>, <another ingress rule...>,...]
}'
```

> **Caution**
>
> Any properties you omit from this request (aside from the ACL `name`) will be reset to defaults. See: *The PUT method* (page 622).

If you *only* want to update the `config` custom user keys, see: *Edit a custom user key via PATCH API* (page 228).

Example

Consider an ACL named `my-acl` with the following properties (shown in JSON):

```
{
  "name": "my-acl",
  "config": {
    "user.my-key": "my-value"
  },
  "description": "My test ACL",
  "egress": [
    {
      "action": "allow",
      "state": "logged"
    }
  ]
  "ingress": [
    {
      "action": "drop",
      "state": "enabled"
    }
  ]
}
```

This query updates that ACL's `egress` rule `state` from `logged` to `enabled`:

```
lxc query --request PUT /1.0/network-acls/my-acl --data '{
  "egress": [
    {
      "action": "allow",
      "state": "enabled"
    }
  ]
}'
```

After the above query is run, `my-acl` contains the following properties:

```
{
  "name": "test",
  "config": {},
  "description": "",
  "egress": [
    {
      "action": "allow",
      "state": "enabled"
    }
  ],
  "ingress": []
}
```

Note that the `description` and `ingress` properties have been reset to defaults because they were not provided in the API request.

To avoid this behavior and preserve the values of any existing properties, you must include them in the `PUT` request along with the updated property:

```
lxc query --request PUT /1.0/network-acls/my-acl --data '{
  "description": "My test ACL",
  "egress": [
    {
      "action": "allow",
      "state": "enabled"
    }
  ],
  "ingress": [
    {
      "action": "drop",
      "state": "enabled"
    }
  ]
}'
```

Edit a custom user key via PATCH API

There's one more way to add or update a custom `config.user.*` key when using the API. Instead of the PUT method shown in the *Edit other properties* (page 226) section above, you can query the `PATCH /1.0/network-acls/{ACL-name}` endpoint:

```
lxc query --request PATCH /1.0/network-acls/{ACL-name} --data '{
  "config": {
    "user.<custom-key-name>": "<custom-key-value>"
  }
}'
```

Caution

Any ACL properties you omit from this request (aside from `config` and `name`) will be reset to defaults.

This PATCH endpoint allows you to add or update custom `config.user.*` keys without affecting other existing `config.user.*` entries. However, this *partial update behavior* (page 622) applies *only* to the `config` property. For the `description`, `egress`, and `ingress` properties, this request behaves like a *PUT request* (page 622): it replaces any provided values and resets any omitted properties to their defaults. Thus, ensure you include any properties you want to keep.

Example

Consider an ACL named `my-acl` with the following properties (shown in JSON):

```
{
  "name": "my-acl",
  "description": "My test ACL",
  "config": {
    "user.my-key1": "1"
  },
}
```

The following query adds a `config.user.my-key2` key with the value of `2`:

```
lxc query --request PATCH /1.0/network-acls/my-acl --data '{
  "config": {
    "user.my-key2": "2"
  }
}'
```

After sending the above request, `my-acl`'s properties are updated to:

```
{
  "name": "my-acl",
  "description": "",
  "config": {
    "user.my-key1": "1",
    "user.my-key2": "2"
  }
}
```

Note that the request *inserted* the new `user.my-key2` key without affecting the pre-existing `user.my-key1` key. Also notice that the `description` property was not sent in the request, and thus was reset to an empty value.

Delete an ACL

You can only delete an ACL that is not *assigned to a NIC or network* (page 230).

CLI

API

To delete an ACL, run:

```
lxc network acl delete <ACL-name>
```

To delete an ACL, query the `DELETE /1.0/network-acls/{ACL-name}` endpoint:

```
lxc query --request DELETE /1.0/network-acls/{ACL-name}
```

Assign an ACL

An ACL is inactive until it is assigned to one of the following targets:

- a *OVN network* (page 587)
- a *Bridge network* (page 573)
- an *OVN NIC type of an instance* (page 462)

To assign an ACL, you must update the `security.acls` option within its target's configuration.

Assigning one or more ACLs to a NIC or network adds a default rule that rejects all unmatched traffic. See *Configure default actions* (page 232) for details.

Assign an ACL to a bridge or OVN network

CLI

API

To set the network's `security.acls`, run the following command. Set the value to a string that contains the ACL name or names you want to add, and comma-separate multiple names:

Set the network's `security.acls` to a string that contains the ACL name or names you want to add. Comma-separate multiple names:

```
lxc network set <network-name> security.acls="<ACL-name>[,<ACL-name>,...]"
```

For more information about using `lxc network set`, see: *How to configure a network* (page 213).

Example

Set the `my-network` network's `security.acls` to contain three ACLs:

```
lxc network set my-network security.acls="my-acl1,my-acl2,my-acl3"
```

To set the network's `security.acls`, query the `PATCH /1.0/networks/{network-name}` endpoint. Set the value to a string that contains the ACL name or names you want to add, and comma-separate multiple names:

```
lxc query --request PATCH /1.0/networks/{network-name} --data '{
  "config": {
    "security.acls": "<ACL-name>[,<ACL-name>,...]"
  }
}'
```

Example

Set the `my-network` network's `security.acls` to contain three ACLs:

```
lxc query --request PATCH /1.0/networks/my-network --data '{
  "config": {
    "security.acls": "my-acl1,my-acl2,my-acl3"
  }
}'
```

Assign an ACL to the OVN NIC of an instance

For NICs (Network Interface Cards), ACLs can only be used with the *OVN NIC type* (page 462).

An NIC is considered a type of instance *device* (page 447). For general information about configuring instance devices, see: *Configure devices* (page 87).

CLI

API

To assign an ACL to an instance's OVN NIC, run:

```
lxc config device set <instance-name> <NIC-name> security.acls="<ACL-name>[,ACL-name,...]"
```

Example

Assign three ACLs to an instance's OVN NIC:

```
lxc config device set my-instance my-ovn-nic security.acls="my-acl1,my-acl2,my-acl3"
```

To assign an ACL to an instance's OVN NIC, query the `PATCH /1.0/instances/{instance-name}` endpoint. Set `security.acls` to a string that contains the ACL name or names you want to add, and comma-separate multiple names:

```
lxc query --request PATCH /1.0/instances/{instance-name} --data '{
  "devices": {
    "<NIC-name>": {
      "network": <network-name>,
      "type": "nic",
      "security.acls": "<ACL-name>[,<ACL-name>,...]",
      <other options>
    }
  }
}'
```

The `type` and `network` options are required in the body (see: *Required device options* (page 89)).

Caution

Patching an instance device's configuration unsets any options for that device omitted from the PATCH request body. For more information, see *Effects of patching device options* (page 89).

Example

For `my-instance`, set its `my-ovn-nic` device's `security.acls` to contain three ACLs:

```
lxc query --request PATCH /1.0/instances/my-instance --data '{
  "devices": {
    "my-ovn-nic": {
      "network": "my-ovn-network",
      "type": "nic",
      "security.acls": "my-acl1,my-acl2,my-acl3"
    }
  }
}'
```

Additional options

To view additional options for the `security.acls` lists, refer to the configuration options for the target network or NIC:

- Bridget network's *`security.acls`* (page 584)
- OVN network's *`security.acls`* (page 592)
- Instance's OVN NIC *`security.acls`* (page 466)

Configure default actions

When one or more ACLs are assigned to a NIC—either directly or through its network—a default reject rule is added to the NIC. This rule rejects all traffic that doesn't match any of the rules in the assigned ACLs.

You can change this behavior with the network- and NIC-level `security.acls.default.ingress.action` and `security.acls.default.egress.action` settings. The NIC-level settings override the network-level settings.

CLI

API

Configure a default action for a network

To set the default action for a network's egress or ingress traffic, run:

```
lxc network set <network-name> security.acls.default.<egress|ingress>.action=
<allow|reject|drop>
```

Example

To set the default action for inbound traffic to `allow` for all instances on the `my-network` network, run:

```
lxc network set my-network security.acls.default.ingress.action=allow
```

Configure a default action for an instance OVN NIC device

To set the default action for an instance OVN NIC's egress or ingress traffic, run:

```
lxc config device set <instance-name> <NIC-name> security.acls.default.
<egress|ingress>.action=<allow|reject|drop>
```

Example

To set the default action for inbound traffic to `allow` for the `my-ovn-nic` device of `my-instance`, run:

```
lxc config device set my-instance my-ovn-nic security.acls.default.ingress.
action=allow
```

Configure a default action for a network

To set the default action for a network's egress or ingress traffic, query the `PATCH /1.0/networks/{network-name}` endpoint:

```
lxc query --request PATCH /1.0/networks/{network-name} --data '{
  "config": {
    "security.acls.default.egress.action": "<allow|reject|drop>",
    "security.acls.default.ingress.action": "<allow|reject|drop>",
  }
}'
```

Example

Set the `my-network` network's default egress action to `allow`:

```
lxc query --request PATCH /1.0/networks/my-network --data '{
  "config": {
    "security.acls.default.egress.action": "allow"
  }
}'
```

Configure a default action for an instance's OVN NIC device

To set the default action for an instance's OVN NIC's traffic, query the `PATCH /1.0/instances/{instance-name}` endpoint:

```
lxc query --request PATCH /1.0/instances/{instance-name} --data '{
  "devices": {
    "<NIC-name>": {
```

(continues on next page)

```
      "network": <network-name>,
      "type": "nic",
      "security.acls.default.<egress|ingress>.action": "<allow|reject|drop>"
      <other-options>
    }
  }
}'
```

The `type` and `network` options are required in the body (see: *Required device options* (page 89)).

> **Caution**
>
> Patching an instance device's configuration unsets any options for that device omitted from the PATCH request body. For more information, see *Effects of patching device options* (page 89).

Example

This request sets the default action for inbound traffic to `allow` for the `my-ovn-nic` device of `my-instance`:

```
lxc query --request PATCH /1.0/instances/my-instance --data '{
  "devices": {
    "my-ovn-nic": {
      "network": "my-network",
      "type": "nic",
      "security.acls.default.ingress.action": "allow"
    }
  }
}'
```

Bridge limitations

When using network ACLs with a bridge network, be aware of the following limitations:

- Unlike OVN ACLs, bridge ACLs apply only at the boundary between the bridge and the LXD host. This means they can enforce network policies only for traffic entering or leaving the host. firewalls (rules controlling traffic between instances on the same bridge) are not supported.
- *ACL groups and network selectors* (page 224) are not supported.
- If you're using the `iptables` firewall driver, you cannot use IP range subjects (such as `192.0.2.1-192.0.2.10`).
- Baseline network service rules are added before ACL rules in their respective INPUT/OUTPUT chains. Because we cannot differentiate between INPUT/OUTPUT and FORWARD traffic after jumping into the ACL chain, ACL rules cannot block these baseline rules.

How to configure network forwards

Note

Network forwards are available for the *OVN network* (page 587) and the *Bridge network* (page 573).

Network forwards allow an external IP address (or specific ports on it) to be forwarded to an internal IP address (or specific ports on it) in the network that the forward belongs to.

This feature can be useful if you have limited external IP addresses and want to share a single external address between multiple instances. In this case, you have two options:

- Forward all traffic from the external address to the internal address of one instance. This method makes it easy to move the traffic destined for the external address to another instance by simply reconfiguring the network forward.
- Forward traffic from different port numbers of the external address to different instances (and optionally different ports on those instances). This method allows to "share" your external IP address and expose more than one instance at a time.

Tip

Network forwards are very similar to using a *proxy device* (page 499) in NAT mode.

The difference is that network forwards are applied on a network level, while a proxy device is added for an instance. In addition, proxy devices can be used to proxy traffic between different connection types (for example, TCP and Unix sockets).

List network forwards

View a list of all forwards configured on a network:

CLI

API

UI

```
lxc network forward list <network_name>
```

Example:

```
lxc network forward list lxdbr0
```

Note

This list displays the listen address of the network forward and its default target address, if set. To view the target addresses for a network forward's ports *set in its port specifications* (page 243), you can *show details about the network forward* (page 236) or *edit the network forward* (page 248).

Query the `/1.0/networks/{networkName}` endpoint to list all forwards for a network.

```
lxc query --request GET /1.0/networks/{networkName}/forwards
```

Example:

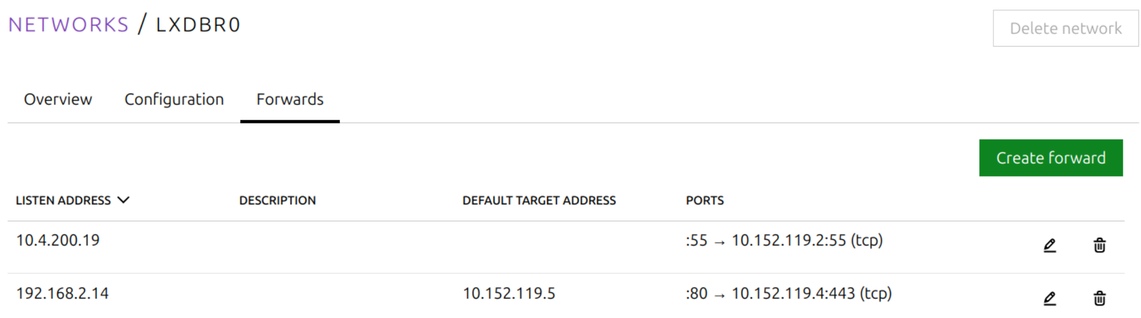
```
lxc query --request GET /1.0/networks/lxdbr0/forwards
```

See the API reference for more information.

You can also use *recursion* (page 621) to list the forwards with a higher level of detail:

```
lxc query --request GET /1.0/networks/{networkName}/forwards?recursion=1
```

In *the web UI* (page 40), select *Networks* in the left sidebar, then select the desired network. On the resulting screen, view the *Forwards* tab:



Show a network forward

Show details about a specific network forward:

CLI

API

UI

```
lxc network forward show <network_name> <listen_address>
```

Example:

```
lxc network forward show lxdbr0 192.0.2.1
```

Query the following endpoint for details about a specific forward:

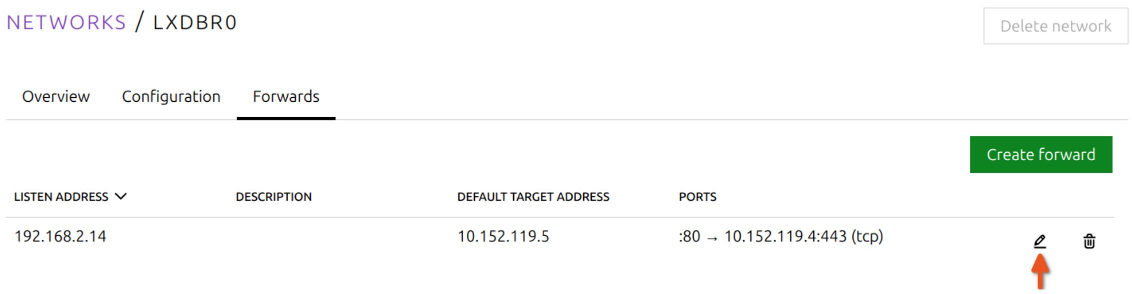
```
lxc query --request GET /1.0/networks/{networkName}/forwards/{listenAddress}
```

See the API reference for more information.

Example:

```
lxc query --request GET /1.0/networks/ovn1/forwards/10.152.119.200
```

In *the web UI* (page 40), select *Networks* in the left sidebar, then select the desired network. On the resulting screen, view the *Forwards* tab. This tab shows you information about all forwards on the network. You can click the *Edit* icon to view details for a specific forward:



Create a network forward

Requirements for listen addresses

Before you can create a network forward, you must understand the requirements for listen addresses.

For both OVN and bridge networks, the listen addresses must not overlap with any subnet in use by other networks on the host. Otherwise, the listen address requirements differ by network type.

OVN network

Bridge network

For an OVN network, the allowed listen addresses must be defined in at least one of the following configuration options, using CIDR notation[119]:

- *`ipv4.routes`* (page 580) or *`ipv6.routes`* (page 583) in the OVN network's uplink network configuration
- *`restricted.networks.subnets`* (page 519) in the OVN network's project configuration

A bridge network does not require you to define allowed listen addresses. Use any non-conflicting IP address available on the host.

Create a forward in an OVN network

> **Note**
>
> You must configure the *allowed listen addresses* (page 237) before you can create a forward in an OVN network.
>
> The IP addresses and ports shown in the examples below are only examples. It is up to you to choose the allowed and available addresses and ports for your setup.

CLI

API

UI

Use the following command to create a forward in an OVN network:

[119] https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing

```
lxc network forward create <ovn_network_name> <listen_address>|--allocate=ipv{4,6}
[target_address=<target_address>] [user.<key>=<value>]
```

- For `<ovn_network_name>`, specify the name of the OVN network on which to create the forward.
- Immediately following the network name, provide only one of the following for the listen address:
  - A listen IP address allowed by the *Requirements for listen addresses* (page 237) (no port number)
  - The `--allocate=` flag with a value of either `ipv4` or `ipv6` for automatic allocation of an allowed IP address
- Optionally provide a default `target_address` (no port number). Any traffic that does not match a port specification is forwarded to this address. This must be an IP address within the OVN network's subnet; typically, the static IP address of an instance is used.
- Optionally provide custom user.* keys to be stored in the network forward's configuration.

Examples

This example shows how to create a network forward on a network named `ovn1` with an allocated listen address and no default target address:

```
lxc network forward create ovn1 --allocate=ipv4
```

This example shows how to create a network forward on a network named `ovn1` with a specific listen address and a default target address:

```
lxc network forward create ovn1 192.0.2.1 target_address=10.41.211.2
```

To create a network forward in an OVN network, send a POST request to the `/1.0/networks/{networkName}/forwards` endpoint:

```
lxc query --request POST /1.0/networks/{networkName}/forwards --data '{
  "listen_address": "<listen_address>",            # required
  "description": "<description of the forward>",   # optional
  "config": {
     "target_address": "<default_target_address>",  # optional
     "user.<key>": "<value>"                        # optional
  },
  "ports": [                                        # optional
    {
      "description": "<description of the forward to this port>",
      "listen_port": "<listen_port>",
      "protocol": "<tcp|udp>",
      "target_address": "<target address>",
      "target_port": "<target port or ports>"
    }
  ]
}'
```

- For {networkName}, specify the name of the OVN network on which to create the forward.
- For <listen_address>, provide only one of the following:
  - A listen IP address allowed by the *Requirements for listen addresses* (page 237) (no port number)
  - For automatic allocation of an allowed IP address, use "0.0.0.0" for IPv4 and "::" for IPv6.
- Optionally provide a description of the forward.
- Optionally provide a default target_address as part of the config object (no port number). Any traffic that does not match a port specification is forwarded to this address. This must be an IP address within the OVN network's subnet; typically, the static IP address of an instance is used.
- Optionally provide custom user.* keys, also as part of the config object.
- Optionally set up port specifications during forward creation. These specifications allow forwarding traffic from specific ports on the listen address to ports on a target address. For details on how to configure ports, see: *Configure ports* (page 243).

See the API reference for more information.

Examples

This example shows how to create a network forward on a network named ovn1 with an allocated listen address and no default target address:

```
lxc query --request POST /1.0/networks/ovn1/forwards --data '{
  "listen_address": "0.0.0.0"
}'
```

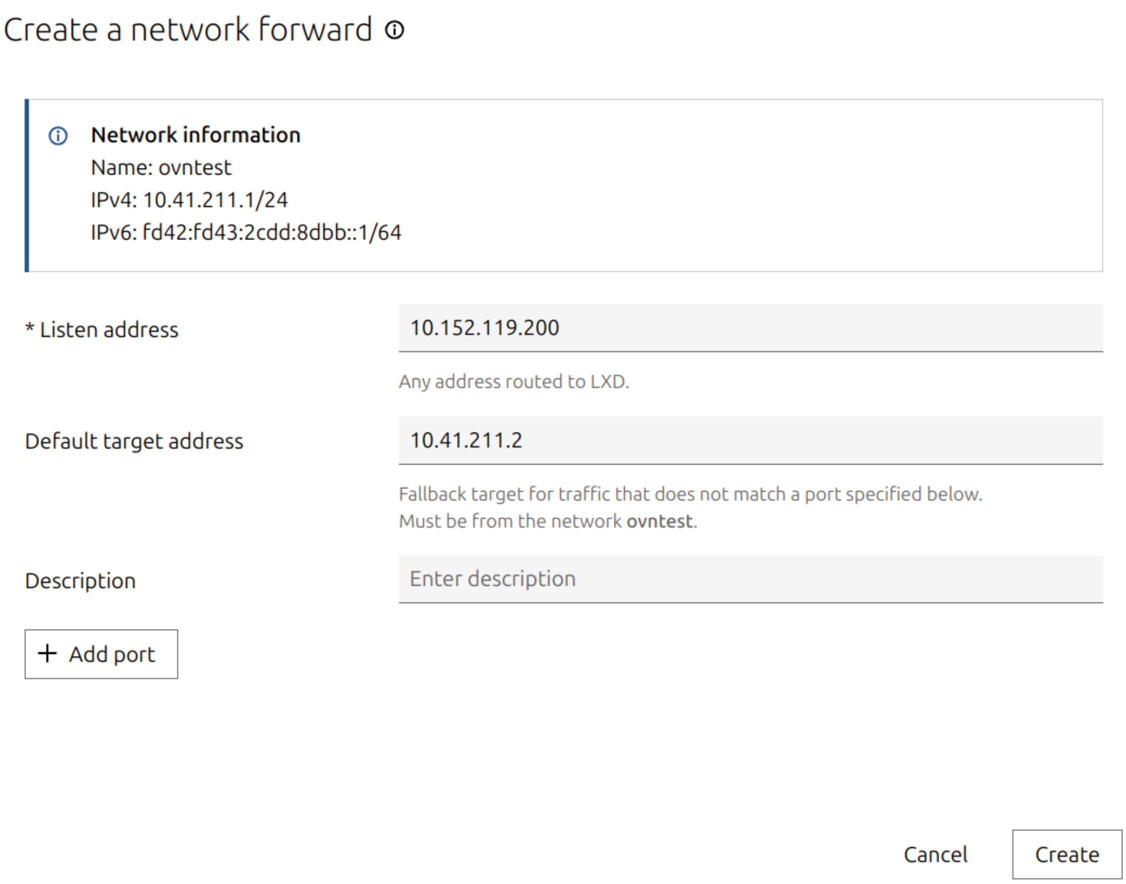
This example shows how to create a network forward on a network named ovn1 with a specific listen address and a default target address:

```
lxc query --request POST /1.0/networks/ovn1/forwards --data '{
  "listen_address": "192.0.2.1",
  "config": {
    "target_address": "10.41.211.2"
  }
}'
```

In *the web UI* (page 40), select *Networks* in the left sidebar, then select the desired OVN network. On the resulting screen, view the *Forwards* tab. Click the *Create forward* button.

In the *Create a new forward* panel, only the *Listen address* field is required.

- For the *Listen address*, provide an IP address allowed by the *Requirements for listen addresses* (page 237) (no port number).
- Optionally provide a *Default target address* (no port number). Any traffic that does not match a port specification is forwarded to this address. This must be an IP address within the OVN network's subnet; typically, the static IP address of an instance is used.



You can optionally set up port specifications for the network forward by clicking the *Add port* button. These specifications allow forwarding traffic from specific ports on the listen address to ports on a target address. For details on how to configure this section, see: *Configure ports* (page 243).

Once you have finished setting up the network forward, click the *Create* button.

Create a forward in a bridge network

> **Note**
>
> The IP addresses and ports shown in the examples below are only examples. It is up to you to choose the allowed and available addresses and ports for your setup.

CLI

API

UI

Use the following command to create a forward in a bridge network:

```
lxc network forward create <bridge_network_name> <listen_address> [target_address=<target_address>] [user.<key>=<value>]
```

- For `<bridge_network_name>`, specify the name of the bridge network on which to create the forward.

- Immediately following the network name, provide an IP address allowed by the *Requirements for listen addresses* (page 237) (no port number).
- Optionally provide a default `target_address` (no port number). Any traffic that does not match a port specification is forwarded to this address. This must be an IP address within the bridge network's subnet; typically, the static IP address of an instance is used.
- Optionally provide custom user.* keys to be stored in the network forward's configuration.
- You cannot use the `--allocate` flag with bridge networks.

Example

This example shows how to create a forward on a network named `bridge1`. The listen address is required, and the default target address is optional:

```
lxc network forward create bridge1 192.0.2.1 target_address=10.41.211.2
```

To create a network forward in a bridge network, send a POST request to the `/1.0/networks/{networkName}/forwards` endpoint:

```
lxc query --request POST /1.0/networks/{networkName}/forwards --data '{
  "listen_address": "<listen_address>",            # required
  "description": "<description of the forward>",   # optional
  "config": {
     "target_address": "<default_target_address>",  # optional
     "user.<key>": "<value>"                        # optional
  },
  "ports": [                                        # optional
    {
      "description": "<description of the forward to this port>",
      "listen_port": "<listen_port>",
      "protocol": "<tcp|udp>",
      "target_address": "<target address>",
      "target_port": "<target port or ports>"
    }
  ]
}'
```

- For `{networkName}`, specify the name of the bridge network on which to create the forward.
- For `<listen_address>`, provide an IP address allowed by the *Requirements for listen addresses* (page 237) (no port number).
  - With bridge networks, you cannot dynamically allocate the listen address. You must input a specific address.
- Optionally provide a description of the forward.
- Optionally provide a default `target_address` as part of the `config` object (no port number). Any traffic that does not match a port specification is forwarded to this address. This must be an IP address within the OVN network's subnet; typically, the static IP address of an instance is used.

- Optionally provide custom `user.*` keys, also as part of the `config` object.
- Optionally set up port specifications during forward creation. These specifications allow forwarding traffic from specific ports on the listen address to ports on a target address. For details on how to configure ports, see: *Configure ports* (page 243).

See the API reference for more information.

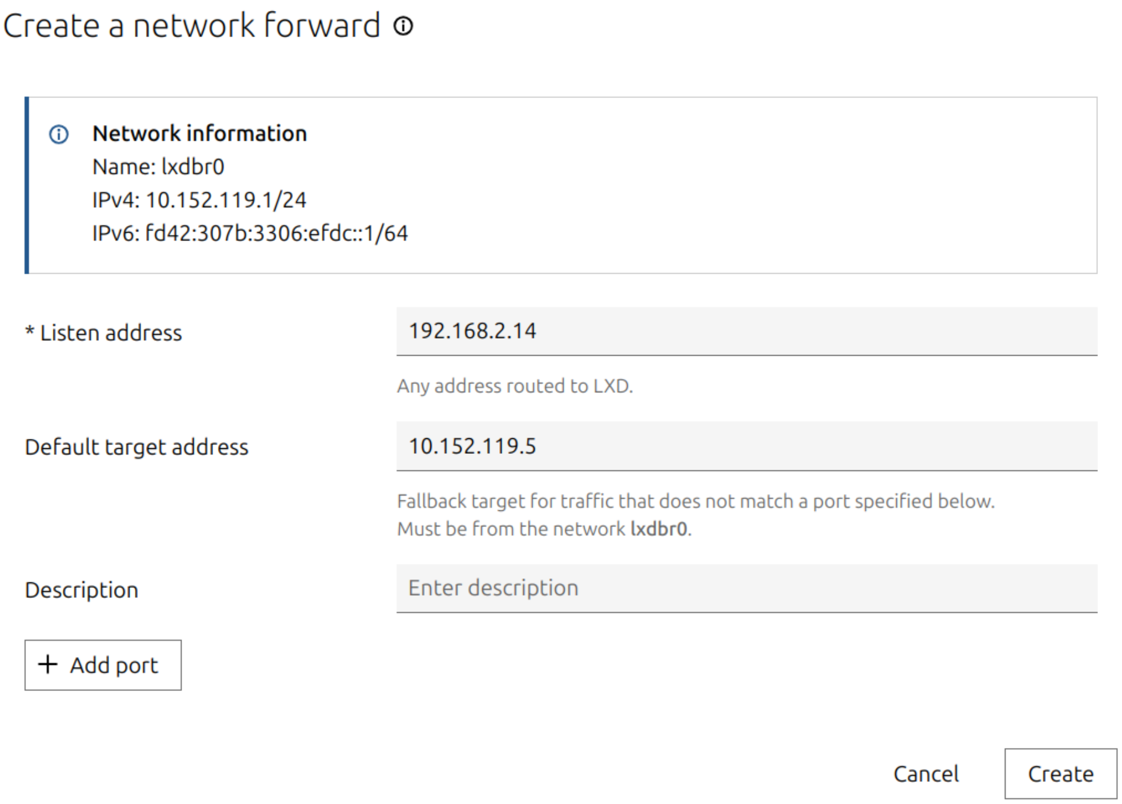
Example

This example shows how to create a forward on a network named `bridge1`. The listen address is required, and the default target address is optional:

```
lxc query --request POST /1.0/networks/bridge1/forwards --data '{
  "listen_address": "192.0.2.1",
  "config": {
    "target_address": "10.41.211.2"
  }
}'
```

In *the web UI* (page 40), select *Networks* in the left sidebar, then select the desired bridge network. On the resulting screen, view the *Forwards* tab. Click the *Create forward* button.

In the *Create a new forward* panel, only the *Listen address* field is required.



- For the *Listen address*, provide a listen IP address allowed by the *Requirements for listen addresses* (page 237) (no port number).
- Optionally provide a *Default target address* (no port number). Any traffic that does not match a port specification is forwarded to this address. This must be an IP address within the bridge network's subnet; typically, the static IP address of an instance is used.

You can optionally set up port specifications for the network forward by clicking the *Add port* button. These specifications allow forwarding traffic from specific ports on the listen address to ports on a target address. For details on how to configure this section, see: *Configure ports* (page 243).

Once you have finished setting up the network forward, click the *Create* button.

Forward properties

Network forwards have the following properties: `config` User-provided free-form key/value pairs (page 243)

Key:	`config`
Type:	string set
Required:	no

The only supported keys are `target_address` and `user.*` custom keys.

The `target_address` key is for the default target address of the network forward. It must be an IP address within the subnet of the network the forward belongs to.

`description` Description of the network forward (page 243)

Key:	`description`
Type:	string
Required:	yes

`listen_address` IP address to listen on (page 243)

Key:	`listen_address`
Type:	string
Required:	no

See *Requirements for listen addresses* (page 237).

`ports` List of port specifications (page 243)

Key:	`ports`
Type:	port list
Required:	no

See *Configure ports* (page 243).

Configure ports

Once a forward is created on a network (whether bridge or OVN), it can be configured with port specifications. These specifications allow forwarding traffic from ports on the listen address to ports on a target address.

CLI

API

UI

When using the CLI, you must first *create a network forward* (page 237) before you can add port specifications to it.

Use the following command to add port specifications on a forward:

```
lxc network forward port add <network_name> <listen_address> <protocol> <listen_
ports> <target_address> [<target_ports>]
```

- Use the network name and listen address of the forward for which you want to add port specifications.
- Use either `tcp` or `udp` as the protocol.
- For the listen ports, you can specify a single listen port, a port range, or a comma-separated set of ports/port ranges.
- Specify a target address. This address must be within the network's subnet, and it must be different from the forward's default target address. Typically, the static IP address of an instance is used.
- Optionally specify a target port or ports. You can:
  - Specify a single target port to forward traffic from all listen ports to this target port.
  - Specify a set of target ports with the same number of set items as the listen ports. This forwards traffic from the first listen port to the first target port, the second listen port to the second target port, and so on.
- If no target port is specified, the listen port value is used for the target port.
- You can add multiple port configurations to the same network forward.

Examples

The example below shows how to configure a forward with a single listen port. Since no target port is specified, the target port defaults to the value of the listen port:

```
lxc network forward port add network1 192.0.2.1 tcp 22 10.41.211.2
```

The example below shows how to configure a forward with a set of listen ports mapped to a single target port. Traffic to the listen address at ports 80 and 90 through 100 is forwarded to port 443 of the target address:

```
lxc network forward port add network1 192.0.2.1 tcp 80,90-100 10.41.211.2 443
```

The example below shows how to configure a forward with a set of listen ports mapped to a set of target ports. Traffic to the listen address at port 22 is forwarded to port 22 of the target address, whereas traffic to port 80 is forwarded to port 443:

```
lxc network forward port add network1 192.0.2.1 tcp 22,80 10.41.211.2 22,443
```

Using the API, you can configure port specifications on a network forward at the time you *create the forward* (page 237), or by *editing the forward* (page 248) after creation.

In either case, you must configure the `ports` object shown below:

```
{
  "listen_address": "<listen_address>",
  "description": "<description of the forward>",
  "config": {
     "target_address": "<default_target_address>",
     "user.<key>": "<value>"
  },
  "ports": [
    {
      "description": "<description of the forward to this port>",
      "listen_port": "<listen_port>",
      "protocol": "<tcp|udp>",
      "target_address": "<target address>",
      "target_port": "<target port or ports>"
    }
  ]
}
```

- For `"listen_port"`, you can specify a single listen port, a port range, or a comma-separated set of ports/port ranges.
- Use either `"tcp"` or `"udp"` as the `"protocol"`.
- Specify a `"target_address"`. This address must be within the network's subnet, and it must be different from the forward's default target address that is configured in the `config` object. Typically, the static IP address of an instance is used.
- Optionally specify a target port or ports. You can:
  - Specify a single target port to forward traffic from all listen ports to this target port.
  - Specify a set of target ports with the same number of set items as the listen ports. This forwards traffic from the first listen port to the first target port, the second listen port to the second target port, and so on.
- If no target port is specified, the listen port value is used for the target port.
- The `"ports"` JSON property is configured as an array (list) of objects. You can set multiple port configurations on the same network forward, each as a separate JSON object in the array.

Examples

```
"ports": [
   {
      "description": "My web server forward",
      "listen_port": "80,81,8080-8090",
      "protocol": "tcp",
      "target_address": "198.51.100.2",
      "target_port": "80,81,8080-8090"
   },
```

(continues on next page)

(continued from previous page)

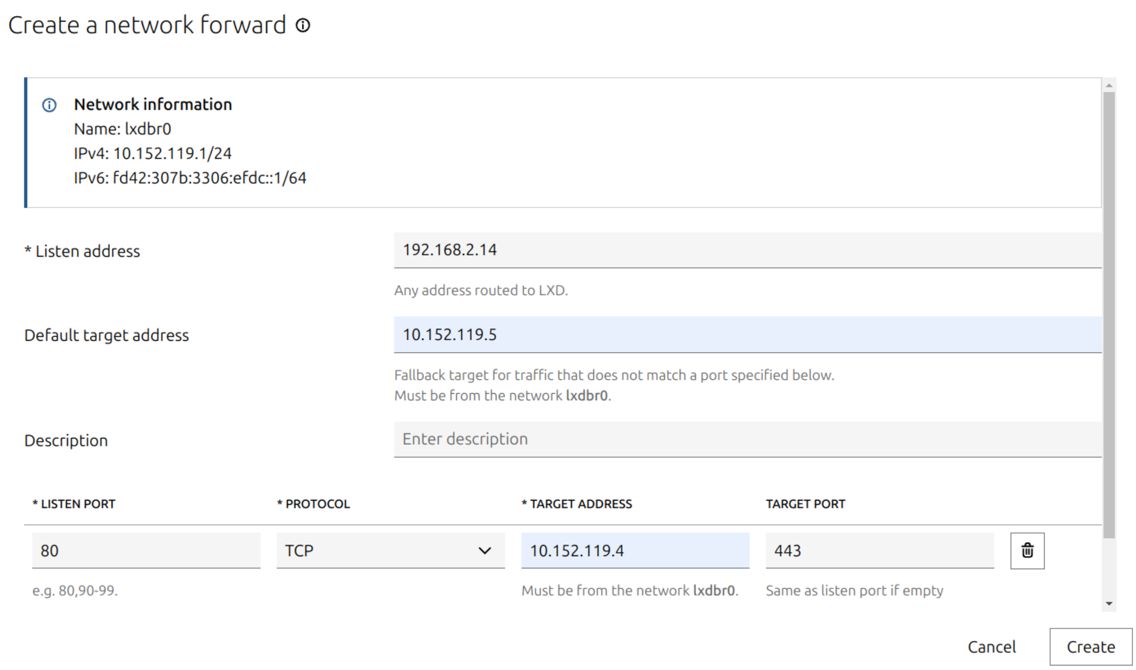
```
    {
        "description": "My API server forward",
        "listen_port": "3000",
        "protocol": "tcp",
        "target_address": "198.51.100.3",
        "target_port": "8080"
    }
]
```

In the example above, traffic to the network forward's listen ports of 80, 81, or 8080-8090 is explicitly forwarded to the same ports on the target address. Traffic to the forward's listen port of 3000 is explicitly forwarded to port 8080 on the target address.

More examples;

- If the `"listen_port"` is set to `"22"` and no `"target_port"` is specified, the target port value defaults to `"22"`.
- If the `"listen_port"` is set to `"80,90-100"` and the `"target_port"` is set to `"442"`, all traffic to the listen address at ports 80 and 90 through 100 is forwarded to port 443 of the target address.
- If the `"listen_port"` is set to `"22,80"` and the `"target_port"` is set to `"22,443"`, all traffic to the listen address at port 22 is forwarded to port 22 of the target address, whereas traffic to port 80 is forwarded to port 443.

In the web UI, you can configure port specifications on a network forward at the time you *create the forward* (page 237), or by *editing the forward* (page 248) after creation.



- For the *Listen port*, you can specify a single port, a port range, or a comma-separated set of ports/port ranges.
- Select either *TCP* or *UDP* as the protocol.

- Specify a *Target address*. This address must be within the network's subnet, and it must be different from the forward's *Default target address*. Typically, the static IP address of an instance is used.
- Optionally specify a target port or ports. You can:
  - Specify a single target port to forward traffic from all listen ports to this target port.
  - Specify a set of target ports with the same number of set items as the listen ports. This forwards traffic from the first listen port to the first target port, the second listen port to the second target port, and so on.
- If no target port is specified, the listen port value is used for the target port.

Examples

- If the *Listen port* is set to `22` and no *Target port* is specified, the target port value defaults to 22.
- If the *Listen port* is set to `80,90-100` and the *Target port* is set to *442*, all traffic to the listen address at ports 80 and 90 through 100 is forwarded to port 443 of the target address.
- If the *Listen port* is set to `22,80` and the *Target port* is set to `22,443`, all traffic to the listen address at port 22 is forwarded to port 22 of the target address, whereas traffic to port 80 is forwarded to port 443.

Port properties

Network forward ports have the following properties: `description` Description of the port or ports (page 247)

Key:	`description`
Type:	string
Required:	no

`listen_port` Listen port or ports (page 247)

Key:	`listen_port`
Type:	string
Required:	yes

For example: `80,90-100`

`protocol` Protocol for the port or ports (page 247)

Key:	`protocol`
Type:	string
Required:	yes

Possible values are `tcp` and `udp`.

`target_address` IP address to forward to (page 247)

Key:	`target_address`
Type:	string
Required:	yes

This `target_address` must be within the subnet of the network the forward belongs to. Also, it must be different from the forward's default target address.

`target_port` Target port or ports (page 248)

Key:	`target_port`
Type:	string
Default:	same as `listen_port`
Required:	no

For example: `70,80-90` or `90`

Edit a network forward

CLI

API

UI

Use the following command to edit a network forward:

```
lxc network forward edit <network_name> <listen_address>
```

This command opens the network forward in YAML format for editing. You can edit both the general configuration and the port specifications.

Partial update

To update a subset of the network forward configuration, send a PATCH request to the `/1.0/networks/{networkName}/forwards/{listenAddress}` endpoint:

```
lxc query --request PATCH /1.0/networks/{networkName}/forwards/{listenAddress} --
data '{
  "config": {
     "target_address": "<default_target_address>",
     "user.<key>": "<value>"
  },
  "description": "<description of the forward>",
  "ports": [
    {
      "description": "<description of the forward to this port>",
      "listen_port": "<listen_port>",
      "protocol": "<tcp|udp>",
      "target_address": "<target address>",
```

(continues on next page)

(continued from previous page)

```
      "target_port": "<target port or ports>"
    }
  ]
}'
```

See the API reference for more information.

Example

Update only the default target address of a forward:

```
lxc query --request PATCH /1.0/networks/ovn1/forwards/10.152.119.200 --data '{
  "config": {
    "target_address": "10.41.211.3"
  }
}'
```

Full update

To replace the entire configuration of an existing network forward, send a PUT request to the `/1.0/networks/{networkName}/forwards/{listenAddress}` endpoint:

```
lxc query --request PUT /1.0/networks/{networkName}/forwards/{listenAddress} --
data '{
  "config": {
     "target_address": "<default_target_address>",
     "user.<key>": "<value>"
  },
  "description": "<description of the forward>",
  "ports": [
    {
      "description": "<description of the forward to this port>",
      "listen_port": "<listen_port>",
      "protocol": "<tcp|udp>",
      "target_address": "<target address>",
      "target_port": "<target port or ports>"
    }
  ]
}'
```

Unlike a `PATCH` request, the `PUT` request replaces the entire configuration.

See the API reference for more information.

Example

When using PUT, take care to send any data should be kept in the configuration. Consider the following configuration for a network forward:

```
{
  "listen_address": "10.152.119.200",
```

(continues on next page)

```
  "config": {
     "target_address": "10.41.211.3",
  },
  "ports": [
    {
      "listen_port": "80",
      "protocol": "tcp",
      "target_address": "10.41.211.4",
      "target_port": "443"
    }
  ]
}'
```

The following PUT request updates the entire configuration:

```
lxc query --request PUT /1.0/networks/ovntest/forwards/10.152.119.200 --data '{
  "ports": [
    {
      "listen_port": "80",
      "protocol": "tcp",
      "target_address": "10.41.211.5",
      "target_port": "443"
    }
  ]
}'
```

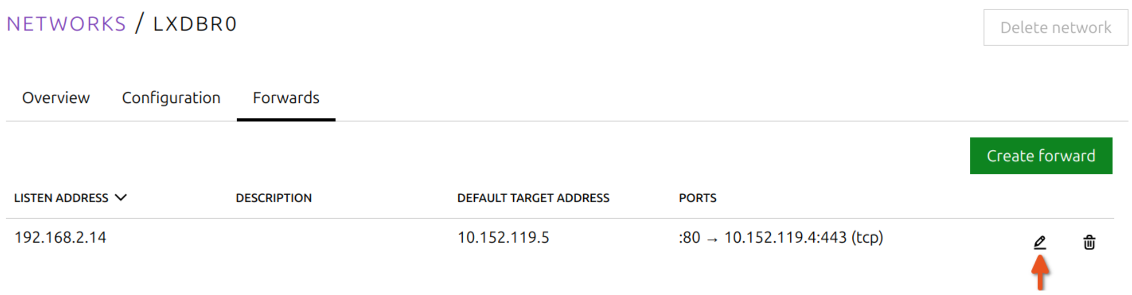
The forward's configuration after the PUT update:

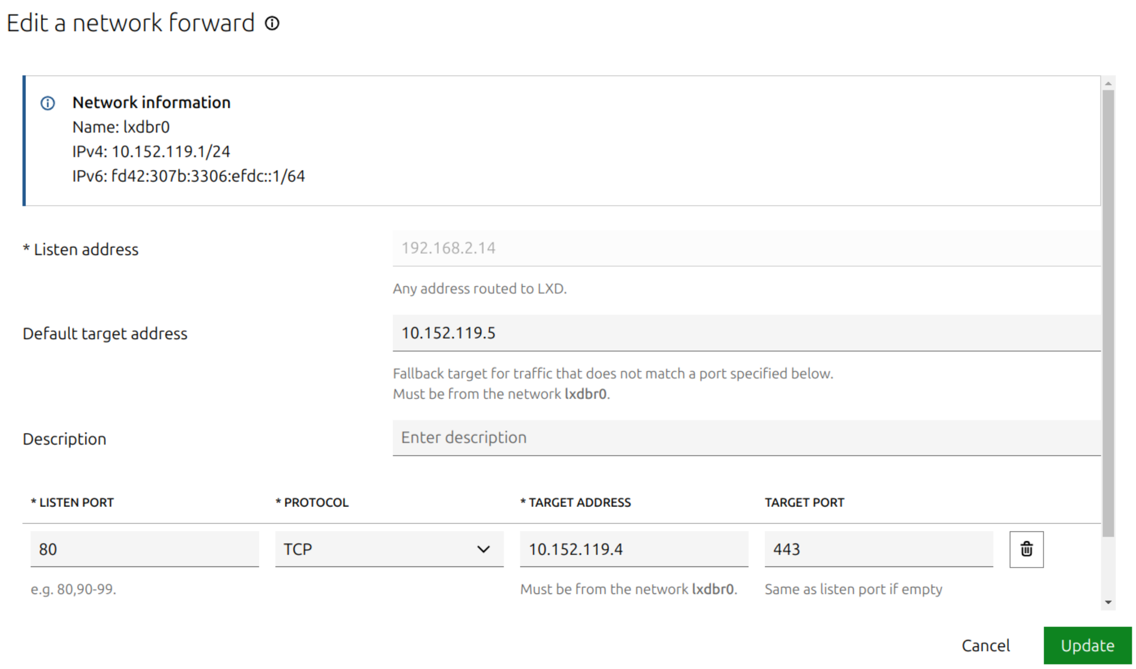
```
{
  "listen_address": "10.152.119.200",
  "config": {},
  "ports": [
    {
      "listen_port": "80",
      "protocol": "tcp",
      "target_address": "10.41.211.5",
      "target_port": "443"
    }
  ]
}
```

Notice that the `config` object no longer contains any values. This is because none were sent as part of the PUT update.

In *the web UI* (page 40), select *Networks* in the left sidebar, then select the desired network. On the resulting screen, view the *Forwards* tab. This tab shows you information about all forwards on the network. Click the *Edit* icon next to a forward to edit it:

In the resulting screen, you can edit the forward's general configuration as well as its port specifications:


NETWORKS / LXDBR0
Delete network
Overview
Configuration
Forwards
Create forward
LISTEN ADDRESS
DESCRIPTION
DEFAULT TARGET ADDRESS
PORTS
192.168.2.14
10.152.119.5
:80 → 10.152.119.4:443 (tcp)


Edit a network forward
Network information
Name: lxdbr0
IPv4: 10.152.119.1/24
IPv6: fd42:307b:3306:efdc::1/64
* Listen address
192.168.2.14
Any address routed to LXD.
Default target address
10.152.119.5
Fallback target for traffic that does not match a port specified below.
Must be from the network lxdbr0.
Description
Enter description
* LISTEN PORT
* PROTOCOL
* TARGET ADDRESS
TARGET PORT
80
TCP
10.152.119.4
443
e.g. 80,90-99.
Must be from the network lxdbr0.
Same as listen port if empty
Cancel
Update

Delete a network forward

CLI

API

UI

Use the following command to delete a network forward:

```
lxc network forward delete <network_name> <listen_address>
```

To delete a network forward, send a DELETE request to the `/1.0/networks/{networkName}/forwards/{listenAddress}` endpoint:

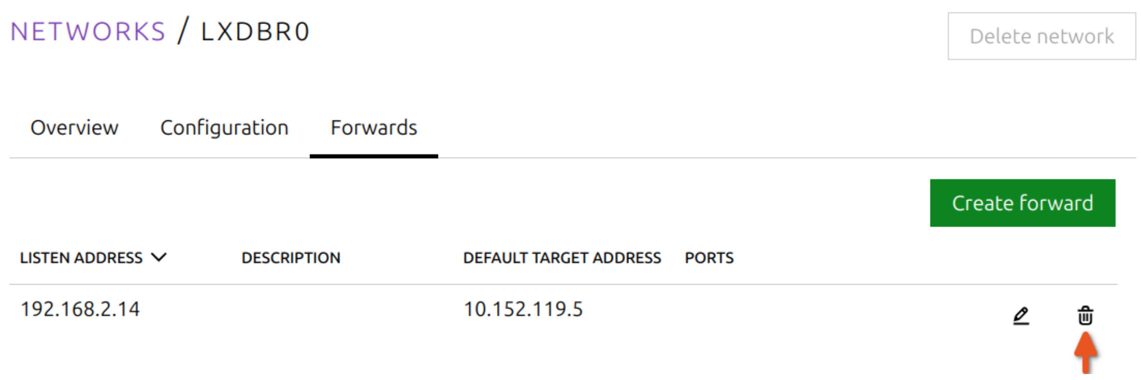
```
lxc query --request DELETE /1.0/networks/{networkName}/forwards/{listenAddress}
```

Example:

```
lxc query --request DELETE /1.0/networks/ovn1/forwards/192.0.2.21
```

See the API reference for more information.

In *the web UI* (page 40), select *Networks* in the left sidebar, then select the desired network. On the resulting screen, view the *Forwards* tab. This tab shows you information about all forwards on the network. Click the *Delete* icon next to a forward to delete it:



How to configure network zones

> **Note**
>
> Network zones are available for the *OVN network* (page 587) and the *Bridge network* (page 573).

Network zones can be used to serve DNS records for LXD networks.

You can use network zones to automatically maintain valid forward and reverse records for all your instances. This can be useful if you are operating a LXD cluster with multiple instances across many networks.

Having DNS records for each instance makes it easier to access network services running on an instance. It is also important when hosting, for example, an outbound SMTP service.

Without correct forward and reverse DNS entries for the instance, sent mail might be flagged as potential spam.

Each network can be associated to different zones:

- Forward DNS records - multiple comma-separated zones (no more than one per project)
- IPv4 reverse DNS records - single zone
- IPv6 reverse DNS records - single zone

LXD will then automatically manage forward and reverse records for all instances, network gateways and downstream network ports and serve those zones for zone transfer to the operator's production DNS servers.

Project views

Projects have a *features.networks.zones* (page 510) feature, which is disabled by default. This controls which project new networks zones are created in. When this feature is enabled new zones are created in the project, otherwise they are created in the default project.

This allows projects that share a network in the default project (i.e those with `features.networks=false`) to have their own project level DNS zones that give a project oriented "view" of the addresses on that shared network (which only includes addresses from instances in their project).

Generated records

Forward records

If you configure a zone with forward DNS records for `lxd.example.net` for your network, it generates records that resolve the following DNS names:

- For all instances in the network: `<instance_name>.lxd.example.net`
- For the network gateway: `<network_name>.gw.lxd.example.net`
- For downstream network ports (for network zones set on an uplink network with a downstream OVN network): `<project_name>-<downstream_network_name>.uplink.lxd.example.net`
- Manual records added to the zone.

You can check the records that are generated with your zone setup with the `dig` command.

This assumes that *core.dns_address* (page 402) was set to `<DNS_server_IP>:<DNS_server_PORT>`. (Setting that configuration option causes the backend to immediately start serving on that address.)

In order for the `dig` request to be allowed for a given zone, you must set the `peers.NAME.address` configuration option for that zone. `NAME` can be anything random. The value must match the IP address where your `dig` is calling from. You must leave `peers.NAME.key` for that same random `NAME` unset.

For example: `lxc network zone set lxd.example.net peers.whatever.address=192.0.2.1`.

Note

It is not enough for the address to be of the same machine that `dig` is calling from; it needs to match as a string with what the DNS server in `lxd` thinks is the exact remote address. `dig` binds to `0.0.0.0`, therefore the address you need is most likely the same that you provided to *core.dns_address* (page 402).

For example, running `dig @<DNS_server_IP> -p <DNS_server_PORT> axfr lxd.example.net` might give the following output:

```
~$ dig @192.0.2.200 -p 1053 axfr lxd.example.net lxd.example.net. 3600 IN SOA
lxd.example.net. ns1.lxd.example.net. 1669736788 120 60 86400
30lxd.example.net. 300 IN NS ns1.lxd.example.net.lxdtest.gw.lxd.example.net.
300 IN A 192.0.2.1lxdtest.gw.lxd.example.net. 300 IN AAAA
fd42:4131:a53c:7211::1default-ovntest.uplink.lxd.example.net. 300 IN A
192.0.2.20default-ovntest.uplink.lxd.example.net. 300 IN AAAA
fd42:4131:a53c:7211:216:3eff:fe4e:b794c1.lxd.example.net. 300 IN AAAA
fd42:4131:a53c:7211:216:3eff:fe19:6edec1.lxd.example.net. 300 IN A
192.0.2.125manualtest.lxd.example.net. 300 IN A 8.8.8.8lxd.example.net. 3600
IN SOA lxd.example.net. ns1.lxd.example.net. 1669736788 120 60 86400 30
```

Reverse records

If you configure a zone for IPv4 reverse DNS records for `2.0.192.in-addr.arpa` for a network using `192.0.2.0/24`, it generates reverse `PTR` DNS records for addresses from all projects that are referencing that network via one of their forward zones.

For example, running `dig @<DNS_server_IP> -p <DNS_server_PORT> axfr 2.0.192.in-addr.arpa` might give the following output:

```
~$ dig @192.0.2.200 -p 1053 axfr 2.0.192.in-addr.arpa 2.0.192.in-addr.arpa.
3600 IN SOA 2.0.192.in-addr.arpa. ns1.2.0.192.in-addr.arpa. 1669736828 120 60
86400 302.0.192.in-addr.arpa. 300 IN NS
ns1.2.0.192.in-addr.arpa.1.2.0.192.in-addr.arpa. 300 IN PTR
lxdtest.gw.lxd.example.net.20.2.0.192.in-addr.arpa. 300 IN PTR
default-ovntest.uplink.lxd.example.net.125.2.0.192.in-addr.arpa. 300 IN PTR
c1.lxd.example.net.2.0.192.in-addr.arpa. 3600 IN SOA 2.0.192.in-addr.arpa.
ns1.2.0.192.in-addr.arpa. 1669736828 120 60 86400 30
```

Enable the built-in DNS server

To make use of network zones, you must enable the built-in DNS server.

To do so, set the *core.dns_address* (page 402) configuration option to a local address on the LXD server. To avoid conflicts with an existing DNS we suggest not using the port 53. This is the address on which the DNS server will listen. Note that in a LXD cluster, the address may be different on each cluster member.

Note

The built-in DNS server supports only zone transfers through AXFR. It cannot be directly queried for DNS records. Therefore, the built-in DNS server must be used in combination with an external DNS server (`bind9`, `nsd`, ...), which will transfer the entire zone from LXD, refresh it upon expiry and provide authoritative answers to DNS requests.

Authentication for zone transfers is configured on a per-zone basis, with peers defined in the zone configuration and a combination of IP address matching and TSIG-key based authentication.

Create and configure a network zone

Use the following command to create a network zone:

```
lxc network zone create <network_zone> [configuration_options...]
```

The following examples show how to configure a zone for forward DNS records, one for IPv4 reverse DNS records and one for IPv6 reverse DNS records, respectively:

```
lxc network zone create lxd.example.net
lxc network zone create 2.0.192.in-addr.arpa
lxc network zone create 1.0.0.0.1.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa
```

Note

Zones must be globally unique, even across projects. If you get a creation error, it might be due to the zone already existing in another project.

You can either specify the configuration options when you create the network or configure them afterwards with the following command:

```
lxc network zone set <network_zone> <key>=<value>
```

Use the following command to edit a network zone in YAML format:

```
lxc network zone edit <network_zone>
```

Configuration options

The following configuration options are available for network zones: `dns.nameservers` Comma-separated list of DNS server FQDNs (for NS records) (page 255)

Key:	`dns.nameservers`
Type:	string set
Required:	no

`network.nat` Whether to generate records for NAT-ed subnets (page 255)

Key:	`network.nat`
Type:	bool
Default:	true
Required:	no

`peers.NAME.address` IP address of a DNS server (page 256)

Key:	`peers.NAME.address`
Type:	string
Required:	no

`peers.NAME.key` TSIG key for the server (page 256)

Key:	`peers.NAME.key`
Type:	string
Required:	no

`user.*` User-provided free-form key/value pairs (page 256)

Key:	`user.*`
Type:	string
Required:	no

Note

When generating the TSIG key using `tsig-keygen`, the key name must follow the format `<zone_name>_<peer_name>.`. For example, if your zone name is `lxd.example.net` and the peer name is `bind9`, then the key name must be `lxd.example.net_bind9.`. If this format is not followed, zone transfer might fail.

Add a network zone to a network

To add a zone to a network, set the corresponding configuration option in the network configuration:

- For forward DNS records: `dns.zone.forward`
- For IPv4 reverse DNS records: `dns.zone.reverse.ipv4`
- For IPv6 reverse DNS records: `dns.zone.reverse.ipv6`

For example:

```
lxc network set <network_name> dns.zone.forward="lxd.example.net"
```

Zones belong to projects and are tied to the `networks` features of projects. You can restrict projects to specific domains and sub-domains through the *restricted.networks.zones* (page 519) project configuration key.

Add custom records

A network zone automatically generates forward and reverse records for all instances, network gateways and downstream network ports. If required, you can manually add custom records to a zone.

To do so, use the *lxc network zone record* (page 835) command.

Create a record

Use the following command to create a record:

```
lxc network zone record create <network_zone> <record_name>
```

This command creates an empty record without entries and adds it to a network zone.

Record properties

Records have the following properties: `config` User-provided free-form key/value pairs (page 257)

Key:	`config`
Type:	string set
Required:	no

The only supported keys are `user.*` custom keys.

`description` Description of the record (page 257)

Key:	`description`
Type:	string
Required:	no

`entries` List of DNS entries (page 257)

Key:	`entries`
Type:	entry list
Required:	no

`name` Unique name of the record (page 257)

Key:	name
Type:	string
Required:	yes

Add or remove entries

To add an entry to the record, use the following command:

```
lxc network zone record entry add <network_zone> <record_name> <type> <value> [--
ttl <TTL>]
```

This command adds a DNS entry with the specified type and value to the record.

For example, to create a dual-stack web server, add a record with two entries similar to the following:

```
lxc network zone record entry add <network_zone> <record_name> A 1.2.3.4
lxc network zone record entry add <network_zone> <record_name> AAAA 1234::1234
```

You can use the `--ttl` flag to set a custom time-to-live (in seconds) for the entry. Otherwise, the default of 300 seconds is used.

You cannot edit an entry (except if you edit the full record with *lxc network zone record edit* (page 837)), but you can delete entries with the following command:

```
lxc network zone record entry remove <network_zone> <record_name> <type> <value>
```

How to configure specific networking features (managed bridge networks only):

How to configure your firewall

> **Important**
>
> This guide applies to managed bridge networks only.

Linux firewalls are based on `netfilter`. LXD uses the same subsystem, which can lead to connectivity issues.

If you run a firewall on your system, you might need to configure it to allow network traffic between the managed LXD bridge and the host. Otherwise, some network functionality (DHCP, DNS and external network access) might not work as expected.

You might also see conflicts between the rules defined by your firewall (or another application) and the firewall rules that LXD adds. For example, your firewall might erase LXD rules if it is started after the LXD daemon, which might interrupt network connectivity to the instance.

`xtables` vs. `nftables`

There are different userspace commands to add rules to `netfilter`: `xtables` (`iptables` for IPv4 and `ip6tables` for IPv6) and `nftables`.

`xtables` provides an ordered list of rules, which might cause issues if multiple systems add and remove entries from the list. `nftables` adds the ability to separate rules into namespaces, which helps to separate rules from different applications. However, if a packet is blocked in one namespace, it is not possible for another namespace to allow it. Therefore, rules in one namespace can still affect rules in another namespace, and firewall applications can still impact LXD network functionality.

If your system supports and uses `nftables`, LXD detects this and switches to `nftables` mode. In this mode, LXD adds its rules into the `nftables`, using its own `nftables` namespace.

Use LXD's firewall

By default, managed LXD bridges add firewall rules to ensure full functionality. If you do not run another firewall on your system, you can let LXD manage its firewall rules.

To enable or disable this behavior, use the `ipv4.firewall` or `ipv6.firewall` *configuration options* (page 574).

Use another firewall

Firewall rules added by other applications might interfere with the firewall rules that LXD adds. Therefore, if you use another firewall, you should disable LXD's firewall rules. You must also configure your firewall to allow network traffic between the instances and the LXD bridge, so that the LXD instances can access the DHCP and DNS server that LXD runs on the host.

See the following sections for instructions on how to disable LXD's firewall rules and how to properly configure `firewalld` and UFW, respectively.

Disable LXD's firewall rules

Run the following commands to prevent LXD from setting firewall rules for a specific network bridge (for example, `lxdbr0`):

```
lxc network set <network_bridge> ipv6.firewall false
lxc network set <network_bridge> ipv4.firewall false
```

`firewalld`: Add the bridge to the trusted zone

To allow traffic to and from the LXD bridge in `firewalld`, add the bridge interface to the `trusted` zone. To do this permanently (so that it persists after a reboot), run the following commands:

```
sudo firewall-cmd --zone=trusted --change-interface=<network_bridge> --permanent
sudo firewall-cmd --reload
```

For example:

```
sudo firewall-cmd --zone=trusted --change-interface=lxdbr0 --permanent
sudo firewall-cmd --reload
```

> **Warning**
>
> The commands given above show a simple example configuration. Depending on your use case, you might need more advanced rules and the example configuration might inadvertently introduce a security risk.

UFW: Add rules for the bridge

If UFW has a rule to drop all unrecognized traffic, it blocks the traffic to and from the LXD bridge. In this case, you must add rules to allow traffic to and from the bridge, as well as allowing traffic forwarded to it.

To do so, run the following commands:

```
sudo ufw allow in on <network_bridge>
sudo ufw route allow in on <network_bridge>
sudo ufw route allow out on <network_bridge>
```

For example:

```
sudo ufw allow in on lxdbr0
sudo ufw route allow in on lxdbr0
sudo ufw route allow out on lxdbr0
```

> **Warning**
>
> The commands given above show a simple example configuration. Depending on your use case, you might need more advanced rules and the example configuration might inadvertently introduce a security risk.
>
> Here's an example for more restrictive firewall rules that limit access from the guests to the host to only DHCP and DNS and allow all outbound connections:

```
# allow the guest to get an IP from the LXD host
sudo ufw allow in on lxdbr0 to any port 67 proto udp
sudo ufw allow in on lxdbr0 to any port 547 proto udp

# allow the guest to resolve host names from the LXD host
sudo ufw allow in on lxdbr0 to any port 53

# allow the guest to have access to outbound connections
CIDR4="$(lxc network get lxdbr0 ipv4.address | sed 's|\.[0-9]\+/|.0/|')"
CIDR6="$(lxc network get lxdbr0 ipv6.address | sed 's|:[0-9]\+/|:/|')"
sudo ufw route allow in on lxdbr0 from "${CIDR4}"
sudo ufw route allow in on lxdbr0 from "${CIDR6}"
```

Prevent connectivity issues with LXD and Docker

Running LXD and Docker on the same host can cause connectivity issues. A common reason for these issues is that Docker sets the global FORWARD policy to `drop`, which prevents LXD from forwarding traffic and thus causes the instances to lose network connectivity. See Docker on a router[120] for detailed information.

There are different ways of working around this problem:

Uninstall Docker

The easiest way to prevent such issues is to uninstall Docker from the system that runs LXD and restart the system. You can run Docker inside a LXD container or virtual machine instead.

See Running Docker inside of a LXD container[121] for detailed information.

Enable IPv4 forwarding

If uninstalling Docker is not an option, enabling IPv4 forwarding before the Docker service starts will prevent Docker from modifying the global FORWARD policy. LXD bridge networks enable this setting normally. However, if LXD starts after Docker, then Docker will already have modified the global FORWARD policy.

> **Warning**
>
> Enabling IPv4 forwarding can cause your Docker container ports to be reachable from any machine on your local network. Depending on your environment, this might be undesirable. See local network container access issue[122] for more information.

To enable IPv4 forwarding before Docker starts, ensure that the following `sysctl` setting is enabled:

```
net.ipv4.conf.all.forwarding=1
```

> **Important**
>
> You must make this setting persistent across host reboots.
>
> One way of doing this is to add a file to the `/etc/sysctl.d/` directory using the following commands:
>
> ```
> echo "net.ipv4.conf.all.forwarding=1" > /etc/sysctl.d/99-forwarding.conf
> systemctl restart systemd-sysctl
> ```

Allow egress network traffic flows

If you do not want the Docker container ports to be potentially reachable from any machine on your local network, you can apply a more complex solution provided by Docker.

[120] https://docs.docker.com/network/packet-filtering-firewalls/#docker-on-a-router
[121] https://www.youtube.com/watch?v=_fCSSEyiGro
[122] https://github.com/moby/moby/issues/14041

Use the following commands to explicitly allow egress network traffic flows from your LXD managed bridge interface:

```
iptables  -I DOCKER-USER -i <network_bridge> -j ACCEPT
ip6tables -I DOCKER-USER -i <network_bridge> -j ACCEPT
iptables  -I DOCKER-USER -o <network_bridge> -m conntrack --ctstate RELATED,
ESTABLISHED -j ACCEPT
ip6tables -I DOCKER-USER -o <network_bridge> -m conntrack --ctstate RELATED,
ESTABLISHED -j ACCEPT
```

For example, if your LXD managed bridge is called `lxdbr0`, you can allow egress traffic to flow using the following commands:

```
iptables  -I DOCKER-USER -i lxdbr0 -j ACCEPT
ip6tables -I DOCKER-USER -i lxdbr0 -j ACCEPT
iptables  -I DOCKER-USER -o lxdbr0 -m conntrack --ctstate RELATED,ESTABLISHED
-j ACCEPT
ip6tables -I DOCKER-USER -o lxdbr0 -m conntrack --ctstate RELATED,ESTABLISHED
-j ACCEPT
```

> **Important**
>
> You must make these firewall rules persistent across host reboots. How to do this depends on your Linux distribution.

How to integrate with `systemd-resolved`

> **Important**
>
> This guide applies to managed bridge networks only.

If the system that runs LXD uses `systemd-resolved` to perform DNS lookups, you should notify `resolved` of the domains that LXD can resolve. To do so, add the DNS servers and domains provided by a LXD network bridge to the `resolved` configuration.

> **Note**
>
> The *`dns.mode`* (page 577) option must be set to `managed` or `dynamic` if you want to use this feature.
>
> Depending on the configured *`dns.domain`* (page 577), you might need to disable DNSSEC in `resolved` to allow for DNS resolution. This can be done through the `DNSSEC` option in `resolved.conf`.

Configure resolved

To add a network bridge to the `resolved` configuration, specify the DNS addresses and domains for the respective bridge.

DNS address

You can use the IPv4 address, the IPv6 address or both. The address must be specified without the subnet netmask.

To retrieve the IPv4 address for the bridge, use the following command:

```
lxc network get <network_bridge> ipv4.address
```

To retrieve the IPv6 address for the bridge, use the following command:

```
lxc network get <network_bridge> ipv6.address
```

DNS domain

To retrieve the DNS domain name for the bridge, use the following command:

```
lxc network get <network_bridge> dns.domain
```

If this option is not set, the default domain name is `lxd`.

Use the following commands to configure `resolved`:

```
resolvectl dns <network_bridge> <dns_address>
resolvectl domain <network_bridge> ~<dns_domain>
```

> **Note**
>
> When configuring `resolved` with the DNS domain name, you should prefix the name with ~. The ~ tells `resolved` to use the respective name server to look up only this domain.
>
> Depending on which shell you use, you might need to include the DNS domain in quotes to prevent the ~ from being expanded.

For example:

```
resolvectl dns lxdbr0 192.0.2.10
resolvectl domain lxdbr0 '~lxd'
```

> **Note**
>
> Alternatively, you can use the `systemd-resolve` command. This command has been deprecated in newer releases of `systemd`, but it is still provided for backwards compatibility.
>
> ```
> systemd-resolve --interface <network_bridge> --set-domain ~<dns_domain> --set-dns <dns_address>
> ```

The `resolved` configuration persists as long as the bridge exists. You must repeat the commands after each reboot and after LXD is restarted, or make it persistent as described below.

Make the resolved configuration persistent

There are two approaches to automating systemd-resolved configuration to ensure that it persists when the LXD bridge network is re-created. Use only one of these approaches, described below.

The first approach is recommended because it is more resilient. It applies your desired configuration whenever your system is rebooted, *and* whenever the LXD bridge network is re-created outside of a system reboot. For example, updating and restarting LXD can occasionally cause its bridge network to be re-created.

If you are unable to use the recommended approach, the alternative approach can be used. The alternative approach applies your desired configuration only when your system is rebooted. If LXD re-creates its bridge network outside of a system reboot, you must reapply the configuration manually.

Recommended approach

Create a systemd network file

Get the network bridge address with the following command:

```
lxc network get lxdbr0 ipv4.address
```

Create a systemd network file named /etc/systemd/network/<network_bridge>.network with the following content:

```
[Match]
Name=<network_bridge>
[Network]
Address=<network_bridge_address>
DNS=<dns_address>
Domains=~<dns_domain>
```

Example file content for /etc/systemd/network/lxdbr0.network (insert your own DNS value):

```
[Match]
Name=lxdbr0
[Network]
Address=10.167.146.1/24
DNS=10.167.146.1
Domains=~lxd
```

Apply the updated configuration

If you have rebooted since you first installed LXD, you only need to reload systemd-resolved:

```
systemctl restart systemd-resolved.service
```

If you have *not* rebooted your system since you first installed LXD, you must either:

1. reboot the system, or
2. reload systemd-networkd (to reload the .network files) and restart lxd (to add the routing):

```
networkctl reload
snap restart lxd
```

You can test that the updated configuration was applied by running:

```
resolvectl status
```

The output should contain a section similar to the example shown below. You should see the configured DNS server and the `~lxd` domain:

```
[...]
Link 4 (lxdbr0)
    Current Scopes: DNS
         Protocols: -DefaultRoute +LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported
Current DNS Server: 10.167.146.1
       DNS Servers: 10.167.146.1
        DNS Domain: ~lxd
[...]
```

Alternative approach

> **Warning**
>
> This approach only automates applying your desired configuration when your system is rebooted. If LXD re-creates its bridge network outside of a system reboot, you must reapply the configuration manually with the following command:
>
> ```
> systemctl restart lxd-dns-<bridge_network>.service
> ```
>
> Example:
>
> ```
> systemctl restart lxd-dns-lxdbr0.service
> ```

Create a `systemd` unit file named `/etc/systemd/system/lxd-dns-<network_bridge>.service` with the following content:

```
[Unit]
Description=LXD per-link DNS configuration for <network_bridge>
BindsTo=sys-subsystem-net-devices-<network_bridge>.device
After=sys-subsystem-net-devices-<network_bridge>.device

[Service]
Type=oneshot
ExecStart=/usr/bin/resolvectl dns <network_bridge> <dns_address>
ExecStart=/usr/bin/resolvectl domain <network_bridge> <dns_domain>
ExecStopPost=/usr/bin/resolvectl revert <network_bridge>
RemainAfterExit=yes

[Install]
WantedBy=sys-subsystem-net-devices-<network_bridge>.device
```

Replace `<network_bridge>` in the file name and content with the name of your bridge (for example, `lxdbr0`). Also replace `<dns_address>` and `<dns_domain>` as described in *Configure resolved* (page 262).

Then enable and start the service with the following commands:

```
sudo systemctl daemon-reload
sudo systemctl enable --now lxd-dns-<network_bridge>
```

If the respective bridge already exists (because LXD is already running), you can use the following command to check that the new service has started:

```
sudo systemctl status lxd-dns-<network_bridge>.service
```

You should see output similar to the following:

```
~$ sudo systemctl status lxd-dns-lxdbr0.service □ lxd-dns-lxdbr0.service -
LXD per-link DNS configuration for lxdbr0 Loaded: loaded
(/etc/systemd/system/lxd-dns-lxdbr0.service; enabled; vendor preset:
enabled) Active: inactive (dead) since Mon 2021-06-14 17:03:12 BST; 1min 2s
ago Process: 9433 ExecStart=/usr/bin/resolvectl dns lxdbr0 n.n.n.n
(code=exited, status=0/SUCCESS) Process: 9434 ExecStart=/usr/bin/resolvectl
domain lxdbr0 ~lxd (code=exited, status=0/SUCCESS) Main PID: 9434
(code=exited, status=0/SUCCESS)
```

To check that `resolved` has applied the settings, use `resolvectl status <network_bridge>`:

```
~$ resolvectl status lxdbr0 Link 6 (lxdbr0) Current Scopes: DNSDefaultRoute
setting: no LLMNR setting: yesMulticastDNS setting: no DNSOverTLS setting:
no DNSSEC setting: no DNSSEC supported: no Current DNS Server: n.n.n.n DNS
Servers: n.n.n.n DNS Domain: ~lxd
```

How to configure specific networking features (OVN networks only):

How to set up OVN with LXD

See the following sections for how to set up a basic OVN network, either as a standalone network or to host a small LXD cluster.

Set up a standalone OVN network

Complete the following steps to create a standalone OVN network that is connected to a managed LXD parent bridge network (for example, `lxdbr0`) for outbound connectivity.

1. Install the OVN tools on the local server:

   ```
   sudo apt install ovn-host ovn-central
   ```

2. Configure the OVN integration bridge:

   ```
   sudo ovs-vsctl set open_vswitch . \
      external_ids:ovn-remote=unix:/var/run/ovn/ovnsb_db.sock \
   ```

(continues on next page)

(continued from previous page)

```
  external_ids:ovn-encap-type=geneve \
  external_ids:ovn-encap-ip=127.0.0.1
```

3. Create an OVN network:

```
lxc network set <parent_network> ipv4.dhcp.ranges=<IP_range> ipv4.ovn.ranges=
<IP_range>
lxc network create ovntest --type=ovn network=<parent_network>
```

4. Create an instance that uses the `ovntest` network:

```
lxc init ubuntu:24.04 c1
lxc config device override c1 eth0 network=ovntest
lxc start c1
```

5. Run *lxc list* (page 785) to show the instance information:

```
~$ lxc list
+------+---------+---------------------+-----------------------------------
NAME | STATE | IPV4 | IPV6 | TYPE | SNAPSHOTS
|+------+---------+---------------------+-----------------------------------
c1 | RUNNING | 192.0.2.2 (eth0) | 2001:db8:cff3:5089:216:3eff:fef0:549f
(eth0) | CONTAINER | 0
|+------+---------+---------------------+-----------------------------------
```

Set up a LXD cluster on OVN

Complete the following steps to set up a LXD cluster that uses an OVN network.

Just like LXD, the distributed database for OVN must be run on a cluster that consists of an odd number of members. The following instructions use the minimum of three servers, which run both the distributed database for OVN and the OVN controller. In addition, you can add any number of servers to the LXD cluster that run only the OVN controller. See the linked YouTube video for the complete tutorial using four machines.

1. Complete the following steps on the three machines that you want to run the distributed database for OVN:

   1. Install the OVN tools:

      ```
      sudo apt install ovn-central ovn-host
      ```

   2. Mark the OVN services as enabled to ensure that they are started when the machine boots:

      ```
       systemctl enable ovn-central
       systemctl enable ovn-host
      ```

   3. Stop OVN for now:

      ```
      systemctl stop ovn-central
      ```

   4. Note down the IP address of the machine:

```
ip -4 a
```

5. Open `/etc/default/ovn-central` for editing.
6. Paste in one of the following configurations (replace `<server_1>`, `<server_2>` and `<server_3>` with the IP addresses of the respective machines, and `<local>` with the IP address of the machine that you are on).
   - For the first machine:

     ```
     OVN_CTL_OPTS=" \
          --db-nb-addr=<local> \
          --db-nb-create-insecure-remote=yes \
          --db-sb-addr=<local> \
          --db-sb-create-insecure-remote=yes \
          --db-nb-cluster-local-addr=<local> \
          --db-sb-cluster-local-addr=<local> \
          --ovn-northd-nb-db=tcp:<server_1>:6641,tcp:<server_2>:6641,tcp:
     <server_3>:6641 \
          --ovn-northd-sb-db=tcp:<server_1>:6642,tcp:<server_2>:6642,tcp:
     <server_3>:6642"
     ```

   - For the second and third machine:

     ```
     OVN_CTL_OPTS=" \
           --db-nb-addr=<local> \
          --db-nb-cluster-remote-addr=<server_1> \
          --db-nb-create-insecure-remote=yes \
          --db-sb-addr=<local> \
          --db-sb-cluster-remote-addr=<server_1> \
          --db-sb-create-insecure-remote=yes \
          --db-nb-cluster-local-addr=<local> \
          --db-sb-cluster-local-addr=<local> \
          --ovn-northd-nb-db=tcp:<server_1>:6641,tcp:<server_2>:6641,tcp:
     <server_3>:6641 \
          --ovn-northd-sb-db=tcp:<server_1>:6642,tcp:<server_2>:6642,tcp:
     <server_3>:6642"
     ```

7. Start OVN:

   ```
   systemctl start ovn-central
   ```

2. On the remaining machines, install only `ovn-host` and make sure it is enabled:

   ```
   sudo apt install ovn-host
   systemctl enable ovn-host
   ```

3. On all machines, configure Open vSwitch (replace the variables as described above):

   ```
   sudo ovs-vsctl set open_vswitch . \
      external_ids:ovn-remote=tcp:<server_1>:6642,tcp:<server_2>:6642,tcp:
   <server_3>:6642 \
   ```

(continues on next page)

(continued from previous page)

```
   external_ids:ovn-encap-type=geneve \
   external_ids:ovn-encap-ip=<local>
```

4. Create a LXD cluster by running `lxd init` on all machines. On the first machine, create the cluster. Then join the other machines with tokens by running *`lxc cluster add <machine_name>`* (page 716) on the first machine and specifying the token when initializing LXD on the other machine.

5. On the first machine, create and configure the uplink network:

```
lxc network create UPLINK --type=physical parent=<uplink_interface> --target=
<machine_name_1>
lxc network create UPLINK --type=physical parent=<uplink_interface> --target=
<machine_name_2>
lxc network create UPLINK --type=physical parent=<uplink_interface> --target=
<machine_name_3>
lxc network create UPLINK --type=physical parent=<uplink_interface> --target=
<machine_name_4>
lxc network create UPLINK --type=physical \
   ipv4.ovn.ranges=<IP_range> \
   ipv6.ovn.ranges=<IP_range> \
   ipv4.gateway=<gateway> \
   ipv6.gateway=<gateway> \
   dns.nameservers=<name_server>
```

   To determine the required values:

   Uplink interface

   A high availability OVN cluster requires a shared layer 2 network, so that the active OVN chassis can move between cluster members (which effectively allows the OVN router's external IP to be reachable from a different host).

   Therefore, you must specify either an unmanaged bridge interface or an unused physical interface as the parent for the physical network that is used for OVN uplink. The instructions assume that you are using a manually created unmanaged bridge. See How to configure network bridges[123] for instructions on how to set up this bridge.

   Gateway

   Run `ip -4 route show default` and `ip -6 route show default`.

   Name server

   Run `resolvectl`.

   IP ranges

   Use suitable IP ranges based on the assigned IPs.

6. Still on the first machine, configure LXD to be able to communicate with the OVN DB cluster. To do so, find the value for `ovn-northd-nb-db` in `/etc/default/ovn-central` and provide it to LXD with the following command:

[123] https://netplan.readthedocs.io/en/stable/examples/#how-to-configure-network-bridges

```
lxc config set network.ovn.northbound_connection <ovn-northd-nb-db>
```

> **Note**
>
> If you are using a MicroOVN deployment, pass the value of the MicroOVN node IP address you want to target. Prefix the IP address with `ssl:`, and suffix it with the `:6641` port number that corresponds to the OVN central service within MicroOVN.

7. Finally, create the actual OVN network (on the first machine):

```
lxc network create my-ovn --type=ovn
```

8. To test the OVN network, create some instances and check the network connectivity:

```
lxc launch ubuntu:24.04 c1 --network my-ovn
lxc launch ubuntu:24.04 c2 --network my-ovn
lxc launch ubuntu:24.04 c3 --network my-ovn
lxc launch ubuntu:24.04 c4 --network my-ovn
lxc list
lxc exec c4 -- bash
ping <IP of c1>
ping <nameserver>
ping6 -n www.example.com
```

Send OVN logs to LXD

Complete the following steps to have the OVN controller send its logs to LXD.

1. Enable the syslog socket:

```
lxc config set core.syslog_socket=true
```

2. Open `/etc/default/ovn-host` for editing.
3. Paste the following configuration:

```
OVN_CTL_OPTS=" \
       --ovn-controller-log='-vsyslog:info --syslog-method=unix:/var/snap/
lxd/common/lxd/syslog.socket'"
```

4. Restart the OVN controller:

```
systemctl restart ovn-controller.service
```

You can now use *`lxc monitor`* (page 787) to see logs from the OVN controller:

```
lxc monitor --type=ovn
```

You can also send the logs to Loki. To do so, add the `ovn` value to the *`loki.types`* (page 411) configuration key, for example:

```
lxc config set loki.types=ovn
```

Tip

You can include logs for OVN `northd`, OVN north-bound `ovsdb-server`, and OVN south-bound `ovsdb-server` as well. To do so, edit `/etc/default/ovn-central`:

```
OVN_CTL_OPTS=" \
   --ovn-northd-log='-vsyslog:info --syslog-method=unix:/var/snap/lxd/common/
lxd/syslog.socket' \
   --ovn-nb-log='-vsyslog:info --syslog-method=unix:/var/snap/lxd/common/lxd/
syslog.socket' \
   --ovn-sb-log='-vsyslog:info --syslog-method=unix:/var/snap/lxd/common/lxd/
syslog.socket'"

sudo systemctl restart ovn-central.service
```

How to configure network load balancers

Note

Network load balancers are currently available for the *OVN network* (page 587).

Network load balancers are similar to forwards in that they allow specific ports on an external IP address to be forwarded to specific ports on internal IP addresses in the network that the load balancer belongs to.

The difference between load balancers and forwards is that load balancers can be used to share ingress traffic between multiple internal backend addresses. This feature can be useful if you have limited external IP addresses or want to share a single external address and ports over multiple instances.

A load balancer is made up of:

- A single external listen IP address.
- One or more named backends consisting of an internal IP and optional port ranges.
- One or more listen port ranges that are configured to forward to one or more named backends.

Create a network load balancer

Use the following command to create a network load balancer:

```
lxc network load-balancer create <network_name> [<listen_address>] [--allocate=ipv
{4,6}] [configuration_options...]
```

Example with a specified listen address:

```
lxc network load-balancer create my-ovn-network 192.0.2.178
```

Example with an allocated listen address:

```
lxc network load-balancer create my-ovn-network --allocate=ipv4
```

Each load balancer is assigned to a network.

Listen addresses are subject to restrictions. If a listen address is not specified, the `--allocate` flag must be provided. See *Requirements for listen addresses* (page 273) for more information about which addresses can be load-balanced, as well as how to use the `--allocate` flag.

Load balancer properties

Network load balancers have the following properties: `backends` List of backend specifications (page 272)

Key:	`backends`
Type:	backend list
Required:	no

See *Configure backends* (page 273).

`config` User-provided free-form key/value pairs (page 272)

Key:	`config`
Type:	string set
Required:	no

The only supported keys are `user.*` custom keys.

`description` Description of the network load balancer (page 272)

Key:	`description`
Type:	string
Required:	no

`listen_address` IP address to listen on (page 272)

Key:	`listen_address`
Type:	string
Required:	no

`ports` List of port specifications (page 272)

Key:	`ports`
Type:	port list
Required:	no

See *Configure ports* (page 274).

Requirements for listen addresses

The following requirements must be met for valid listen addresses:

- Allowed listen addresses must be defined in the uplink network's `ipv{n}.routes` settings or the project's *restricted.networks.subnets* (page 519) setting.
  - If you specify a listen address when creating a load balancer, it must be within the range of allowed addresses.
  - If you do not specify a listen address, you must use either `--allocate ipv4` or `--allocate ipv6`. This will allocate a listen address from the range of allowed addresses.
- The listen address must not overlap with a subnet that is in use with another network or entity in that network.

Configure backends

You can add backend specifications to the network load balancer to define target addresses (and optionally ports). The backend target address must be within the same subnet as the network associated with the load balancer.

Use the following command to add a backend specification:

```
lxc network load-balancer backend add <network_name> <listen_address> <backend_
name> <target_address> [<target_ports>]
```

Example:

```
lxc network load-balancer backend add my-ovn-network 192.0.2.178 test-backend 10.
41.211.5
```

If no target ports are specified when adding the backend:

- The load balancer uses the listen ports defined in the *port specification* (page 274) associated with that backend, if any.
- If no such listen ports are defined, the backend has no target ports and is inactive. You must either *add a port specification* (page 274) or *edit the load balancer configuration* (page 275) to include a `target_port` value in the backend specification or a `listen_port` value in the ports specification.

If you want to forward the traffic to different ports, you have two options:

- Specify a single target port to forward traffic from all listen ports to this target port.
- Specify a set of target ports with the same number of ports as the listen ports to forward traffic from the first listen port to the first target port, the second listen port to the second target port, and so on.

Backend properties

Network load balancer backends have the following properties: `description` Description of the backend (page 273)

Key:	`description`
Type:	string
Required:	no

`name` Name of the backend (page 274)

Key:	`name`
Type:	string
Required:	yes

`target_address` IP address to forward to (page 274)

Key:	`target_address`
Type:	string
Required:	yes

`target_port` Target port or ports (page 274)

Key:	`target_port`
Type:	string
Default:	same as *`listen_port`* (page 275)
Required:	no

For example: `70,80-90` or `90`

Configure ports

You can add port specifications to the network load balancer to forward traffic from specific ports on the listen address to specific ports on one or more target backends.

Use the following command to add a port specification:

```
lxc network load-balancer port add <network_name> <listen_address> <protocol>
<listen_ports> <backend_name>[,<backend_name>...]
```

Example:

```
lxc network load-balancer port add my-ovn-network 192.0.2.178 tcp 80 test-backend
```

You can specify a single listen port or a set of ports. The backend(s) specified must have target port(s) settings compatible with the port's listen port(s) setting.

Port properties

Network load balancer ports have the following properties: `description` Description of the port or ports (page 274)

Key:	description
Type:	string
Required:	no

listen_port Listen port or ports (page 275)

Key:	listen_port
Type:	string
Required:	yes

For example: 80,90-100

protocol Protocol for the port or ports (page 275)

Key:	protocol
Type:	string
Required:	yes

Possible values are tcp and udp.

target_backend Backend name or names to forward to (page 275)

Key:	target_backend
Type:	backend list
Required:	yes

Edit a network load balancer

Use the following command to edit a network load balancer:

```
lxc network load-balancer edit <network_name> <listen_address>
```

This command opens the network load balancer in YAML format for editing. You can edit the general configuration, as well as the backend and port specifications.

Example load balancer configuration YAML file:

```yaml
listen_address: 192.0.2.178
location: ""
description: ""
config: {}
backends:
- name: test-backend
  description: ""
  target_port: ""
  target_address: 10.41.211.5
ports:
- description: ""
```

(continues on next page)

(continued from previous page)

```
protocol: tcp
listen_port: 70,80-90
target_backend:
- test-backend
```

Delete a network load balancer

Use the following command to delete a network load balancer:

```
lxc network load-balancer delete <network_name> <listen_address>
```

How to create OVN peer routing relationships

Important

This guide applies to OVN networks only.

By default, traffic between two OVN networks goes through the uplink network. This path is inefficient, however, because packets must leave the OVN subsystem and transit through the host's networking stack (and, potentially, an external network) and back into the OVN subsystem of the target network. Depending on how the host's networking is configured, this might limit the available bandwidth (if the OVN overlay network is on a higher bandwidth network than the host's external network).

Therefore, LXD allows creating peer routing relationships between two OVN networks. Using this method, traffic between the two networks can go directly from one OVN network to the other and thus stays within the OVN subsystem, rather than transiting through the uplink network.

Create a routing relationship between networks

To add a peer routing relationship between two networks, you must create a network peering for both networks. The relationship must be mutual. If you set it up on only one network, the routing relationship will be in pending state, but not active.

When creating the peer routing relationship, specify a peering name that identifies the relationship for the respective network. The name can be chosen freely, and you can use it later to edit or delete the relationship.

Use the following commands to create a peer routing relationship between networks in the same project:

```
lxc network peer create <network1> <peering_name> <network2> [configuration_
options]
lxc network peer create <network2> <peering_name> <network1> [configuration_
options]
```

You can also create peer routing relationships between OVN networks in different projects:

```
lxc network peer create <network1> <peering_name> <project2/network2>
[configuration_options] --project=<project1>
lxc network peer create <network2> <peering_name> <project1/network1>
[configuration_options] --project=<project2>
```

Important

If the project or the network name is incorrect, the command will not return any error indicating that the respective project/network does not exist, and the routing relationship will remain in pending state. This behavior prevents users in a different project from discovering whether a project and network exists.

Peering properties

Peer routing relationships have the following properties: `config` User-provided free-form key/value pairs (page 277)

Key:	`config`
Type:	string set
Required:	no

The only supported keys are `user.*` custom keys.

`description` Description of the network peering (page 277)

Key:	`description`
Type:	string
Required:	no

`name` Name of the network peering on the local network (page 277)

Key:	`name`
Type:	string
Required:	yes

`status` Status indicating if pending or created (page 277)

Key:	`status`
Type:	string
Required:	–

Indicates if mutual peering exists with the target network. This property is read-only and cannot be updated.

`target_network` Which network to create a peering with (page 277)

Key:	`target_network`
Type:	string
Required:	yes

This option must be set at create time.

`target_project` Which project the target network exists in (page 278)

Key:	`target_project`
Type:	string
Required:	yes

This option must be set at create time.

List routing relationships

To list all network peerings for a network, use the following command:

```
lxc network peer list <network>
```

Edit a routing relationship

Use the following command to edit a network peering:

```
lxc network peer edit <network> <peering_name>
```

This command opens the network peering in YAML format for editing.

How to troubleshoot your networking setup:

How to display IPAM information of a LXD deployment

IPAM (IP Address Management) is a method used to plan, track, and manage the information associated with a computer network's IP address space. In essence, it's a way of organizing, monitoring, and manipulating the IP space in a network.

Checking the IPAM information for your LXD setup can help you debug networking issues. You can see which IP addresses are used for instances, network interfaces, forwards, and load balancers and use this information to track down where traffic is lost.

To display IPAM information, enter the following command:

```
lxc network list-allocations
```

By default, this command shows the IPAM information for the `default` project. You can select a different project with the `--project` flag, or specify `--all-projects` to display the information for all projects.

The resulting output will look something like this:

```
+-----------------------+-----------------+----------+------+-------------------+
|        USED BY        |     ADDRESS     |   TYPE   | NAT  | HARDWARE ADDRESS  |
+-----------------------+-----------------+----------+------+-------------------+
| /1.0/networks/lxdbr0  | 192.0.2.0/24    | network  | true |                   |
+-----------------------+-----------------+----------+------+-------------------+
| /1.0/networks/lxdbr0  | 2001:db8::/32   | network  | true |                   |
+-----------------------+-----------------+----------+------+-------------------+
| /1.0/instances/u1     | 2001:db8::2/128 | instance | true | 00:16:3e:04:f0:95 |
+-----------------------+-----------------+----------+------+-------------------+
| /1.0/instances/u1     | 192.0.2.2/32    | instance | true | 00:16:3e:04:f0:95 |
+-----------------------+-----------------+----------+------+-------------------+
```

Each listed entry lists the IP address (in CIDR notation) of one of the following LXD entities: `network`, `network-forward`, `network-load-balancer`, and `instance`. An entry contains an IP address using the CIDR notation. It also contains a LXD resource URI, the type of the entity, whether it is in NAT mode, and the hardware address (only for the `instance` entity).

View DHCP leases for fully controlled networks

LXD can provide the currently held DHCP leases for *fully controlled networks* (page 354):

```
lxc network list-leases <network_name>
```

For example, using `lxdbr0` from above:

```
+-----------+-------------------+-------------+---------+
| HOSTNAME  |    MAC ADDRESS    | IP ADDRESS  |  TYPE   |
+-----------+-------------------+-------------+---------+
| lxdbr0.gw |                   | 192.0.2.1   | GATEWAY |
+-----------+-------------------+-------------+---------+
| lxdbr0.gw |                   | 2001:db8::1 | GATEWAY |
+-----------+-------------------+-------------+---------+
| u1        | 00:16:3e:04:f0:95 | 192.0.2.2   | DYNAMIC |
+-----------+-------------------+-------------+---------+
| u1        | 00:16:3e:04:f0:95 | 2001:db8::2 | DYNAMIC |
+-----------+-------------------+-------------+---------+
```

Related topics

Explanation:

- *Networking setups* (page 353)

Reference:

- *Networks* (page 573)

2.3. Get ready for production

Once you are ready for production, consider setting up a LXD cluster to support the required load. You should also monitor your server or servers and configure them for the expected load.

2.3.1. Clustering

The following how-to guides cover common operations related to clustering.

How to create and configure a cluster:

How to form a cluster

When forming a LXD cluster, you start with a bootstrap server. This bootstrap server can be an existing LXD server or a newly installed one.

After initializing the bootstrap server, you can join additional servers to the cluster. See *Cluster members* (page 370) for more information.

You can form the LXD cluster interactively by providing configuration information during the initialization process or by using preseed files that contain the full configuration.

To quickly and automatically set up a basic LXD cluster, you can use *MicroCloud* (page 284).

Configure the cluster interactively

To form your cluster, you must first run `lxd init` on the bootstrap server. After that, run it on the other servers that you want to join to the cluster.

When forming a cluster interactively, you answer the questions that `lxd init` prompts you with to configure the cluster.

Initialize the bootstrap server

To initialize the bootstrap server, run `lxd init` and answer the questions according to your desired configuration.

You can accept the default values for most questions, but make sure to answer the following questions accordingly:

- `Would you like to use LXD clustering?`

  Select **yes**.

- `What IP address or DNS name should be used to reach this server?`

  Make sure to use an IP or DNS address that other servers can reach.

- `Are you joining an existing cluster?`

  Select **no**.

```
~$ lxd init Would you like to use LXD clustering? (yes/no) [default=no]:
yesWhat IP address or DNS name should be used to reach this server?
[default=192.0.2.101]:Are you joining an existing cluster? (yes/no)
[default=no]: noWhat member name should be used to identify this server in
the cluster? [default=server1]:Do you want to configure a new local storage
pool? (yes/no) [default=yes]:Name of the storage backend to use (btrfs, dir,
lvm, zfs) [default=zfs]:Create a new ZFS pool? (yes/no) [default=yes]:Would
you like to use an existing empty block device (e.g. a disk or partition)?
(yes/no) [default=no]:Size in GiB of the new loop device (1GiB minimum)
[default=9GiB]:Do you want to configure a new remote storage pool? (yes/no)
[default=no]:Would you like to connect to a MAAS server? (yes/no)
```

```
[default=no]:Would you like to configure LXD to use an existing bridge or
host interface? (yes/no) [default=no]:Would you like to create a new Fan
overlay network? (yes/no) [default=yes]:What subnet should be used as the Fan
underlay? [default=auto]:Would you like stale cached images to be updated
automatically? (yes/no) [default=yes]:Would you like a YAML "lxd init"
preseed to be printed? (yes/no) [default=no]:
```

After the initialization process finishes, your first cluster member should be up and available on your network. You can check this with *lxc cluster list* (page 726).

Join additional servers

You can now join further servers to the cluster.

> **Note**
>
> The servers that you add should be newly installed LXD servers. If you are using existing servers, make sure to clear their contents before joining them, because any existing data on them will be lost.

To join a server to the cluster, run `lxd init` on the cluster. Joining an existing cluster requires root privileges, so make sure to run the command as root or with `sudo`.

Basically, the initialization process consists of the following steps:

1. Request to join an existing cluster.

   Answer the first questions that `lxd init` asks accordingly:

   - `Would you like to use LXD clustering?`

     Select **yes**.

   - `What IP address or DNS name should be used to reach this server?`

     Make sure to use an IP or DNS address that other servers can reach.

   - `Are you joining an existing cluster?`

     Select **yes**.

2. Authenticate with the cluster.

   Generate a cluster join token for each new member. To do so, run the following command on an existing cluster member (for example, the bootstrap server):

   ```
   lxc cluster add <new_member_name>
   ```

   This command returns a single-use join token that is valid for a configurable time (see *cluster.join_token_expiry* (page 408)). Enter this token when `lxd init` prompts you for the join token.

   The join token contains the addresses of the existing online members, as well as a single-use secret and the fingerprint of the cluster certificate. This reduces the amount of questions that you must answer during `lxd init`, because the join token can be used to answer these questions automatically.

3. Confirm that all local data for the server is lost when joining a cluster.
4. Configure server-specific settings (see *Member configuration* (page 372) for more information).

   You can accept the default values or specify custom values for each server.

```
~$ sudo lxd init Would you like to use LXD clustering? (yes/no) [default=no]:
yesWhat IP address or DNS name should be used to reach this server?
[default=192.0.2.102]:Are you joining an existing cluster? (yes/no)
[default=no]: yesDo you have a
join token? (yes/no/[token]) [default=no]: yesPlease provide join token: eyJz-
ZXJ2ZXJfbmFtZSI6InJwaTAxIiwiZmluZ2VycHJpbnQiOiIyNjZjZmExZDk0ZDZiMjk2Nzk0YjU0YzJl
existing data is lost when joining a cluster, continue? (yes/no) [default=no]
yesChoose "size" property for storage pool "local":Choose "source" property
for storage pool "local":Choose "zfs.pool_name" property for storage pool
"local":Would you like a YAML "lxd init" preseed to be printed? (yes/no)
[default=no]:
```

After the initialization process finishes, your server is added as a new cluster member. You can check this with *lxc cluster list* (page 726).

Configure the cluster through preseed files

To form your cluster, you must first run `lxd init` on the bootstrap server. After that, run it on the other servers that you want to join to the cluster.

Instead of answering the `lxd init` questions interactively, you can provide the required information through preseed files. You can feed a file to `lxd init` with the following command:

```
cat <preseed-file> | lxd init --preseed
```

You need a different preseed file for every server.

Initialize the bootstrap server

To enable clustering, the preseed file for the bootstrap server must contain the following fields:

```yaml
config:
  core.https_address: <IP_address_and_port>
cluster:
  server_name: <server_name>
  enabled: true
```

Here is an example preseed file for the bootstrap server:

```yaml
config:
  core.https_address: 192.0.2.101:8443
  images.auto_update_interval: 15
storage_pools:
- name: default
  driver: dir
```

(continues on next page)

(continued from previous page)

```yaml
- name: my-pool
  driver: zfs
networks:
- name: lxdbr0
  type: bridge
profiles:
- name: default
  devices:
    root:
      path: /
      pool: my-pool
      type: disk
    eth0:
      name: eth0
      nictype: bridged
      parent: lxdbr0
      type: nic
cluster:
  server_name: server1
  enabled: true
```

See *Preseed YAML file fields* (page 508) for the complete fields of the preseed YAML file.

Join additional servers

The preseed files for new cluster members require only a `cluster` section with data and configuration values that are specific to the joining server.

The preseed file for additional servers must include the following fields:

```yaml
cluster:
  enabled: true
  server_address: <IP_address_of_server>
  cluster_token: <join_token>
```

Here is an example preseed file for a new cluster member:

```yaml
cluster:
  enabled: true
  server_address: 192.0.2.102:8443
  cluster_token:
eyJzZXJ2ZXJfbmFtZSI6Im5vZGUyIiwiZmluZ2VycHJpbnQiOiJjZjlmNmVhMWIzYjhiNjgxNzQ1YTY1NTY2YjM3ZGUwOTUz
  member_config:
  - entity: storage-pool
    name: default
    key: source
    value: ""
  - entity: storage-pool
    name: my-pool
    key: source
```

(continues on next page)

(continued from previous page)

```yaml
    value: ""
  - entity: storage-pool
    name: my-pool
    key: driver
    value: "zfs"
```

See *Preseed YAML file fields* (page 508) for the complete fields of the preseed YAML file.

Use MicroCloud

Instead of setting up your LXD cluster manually, you can use MicroCloud[124] to get a fully highly available LXD cluster with OVN and with Ceph storage up and running.

To install the required snaps, run the following command:

```
snap install lxd microceph microovn microcloud
```

Then start the bootstrapping process with the following command:

```
microcloud init
```

If you want to set up a multi-machine MicroCloud, run the following command on all the other machines:

```
microcloud join
```

Following the CLI prompts, a working MicroCloud will be ready within minutes.

When the initialization is complete, you'll have an OVN cluster, a Ceph cluster and a LXD cluster, and LXD itself will have been configured with both networking and storage suitable for use in a cluster.

See the MicroCloud documentation[125] for more information.

How to manage a cluster

After your cluster is formed, use `lxc cluster list` (page 726) to see a list of its members and their status:

```
~$ lxc cluster list
+---------+--------------------------+-----------------+--------------+-----
NAME | URL | ROLES | ARCHITECTURE | FAILURE DOMAIN | DESCRIPTION | STATE |
MESSAGE
|+---------+--------------------------+-----------------+--------------+-----
server1 | https://192.0.2.101:8443 | database-leader | x86_64 | default | |
ONLINE | Fully operational || | | database | | | | |
|+---------+--------------------------+-----------------+--------------+-----
server2 | https://192.0.
2.102:8443 | database-standby | aarch64 | default | | ONLINE | Fully operational
|+---------+--------------------------+-----------------+--------------+-----
```

[124] https://canonical.com/microcloud
[125] https://documentation.ubuntu.com/microcloud/latest/microcloud/

```
server3 | https://192.0.
2.103:8443 | database-standby | aarch64 | default | | ONLINE | Fully operational
|+---------+-------------------------+----------------+-------------+-----
```

To see more detailed information about an individual cluster member, run the following command:

```
lxc cluster show <member_name>
```

To see state and usage information for a cluster member, run the following command:

```
lxc cluster info <member_name>
```

Configure your cluster

To configure your cluster, use *`lxc config`* (page 738):

```
lxc config set <server-config-option> <value>
```

Example:

```
lxc config set cluster.max_voters 5
```

All LXD *server configuration options* (page 401) can be applied to cluster members.

Keep in mind that some options are global in scope, and others are local. When you configure an option with global scope on any cluster member, the changes are propagated to the other cluster members through the distributed database. The locally scoped options are set only on the cluster member where you configure them, unless you use the `--target` flag to specify a different cluster member.

In addition to the server configuration, there are *cluster member configuration options* (page 602) that are specific to each cluster member. To set these configuration values, use *`lxc cluster set`* (page 731):

```
lxc cluster set <member-name> <member-config-option> <value>
```

Example:

```
lxc cluster set server1 scheduler.instance manual
```

Alternatively, you can use the *use the edit command* (page 286).

Assign member roles

To add or remove a *member role* (page 370) for a cluster member, use the *`lxc cluster role`* (page 729) command:

```
lxc cluster role add <member-name> <role>
```

Example:

```
lxc cluster role add server1 event-hub
```

> **Note**
>
> You can add or remove only those roles that are not assigned automatically by LXD. To find out which roles are automatically assigned, see: *Member roles* (page 370).

Edit the cluster member configuration

To edit all properties of a cluster member, including the member-specific configuration, the member roles, the failure domain and the cluster groups, use the following command:

```
lxc cluster edit
```

For more information, see: `lxc cluster edit` (page 717).

Evacuate and restore cluster members

There are scenarios where you might need to empty a given cluster member of all its instances (for example, for routine maintenance like applying system updates that require a reboot, or to perform hardware changes). The *evacuate* (page 286) and *restore* (page 286) commands simplify this process.

Evacuate a cluster member

The evacuation process migrates all instances on a given cluster member to other members in its cluster. The given member is then set to an "evacuated" state, which prevents the creation of any instances on it.

To begin this process, use the `lxc cluster evacuate` (page 718) command:

```
lxc cluster evacuate <member_name>
```

Restore an evacuated cluster member

When the evacuated cluster member is available again, use the `lxc cluster restore` (page 728) command to return it to a normal running state:

```
lxc cluster restore <member_name>
```

This command removes the cluster member's "evacuated" state, migrates the evacuated instances back from the cluster members that were temporarily holding them (using live migration if applicable), then restarts any instances that were shut down.

Evacuation mode and live migration

You can control how each instance is migrated, via the `cluster.evacuate` (page 416) instance configuration key. This key applies to the migrations performed during both evacuation and restoration. By default, any instances that are suitable for *live migration* (page 136) will be live-migrated, and any that are not suitable will be shut down. See the `cluster.evacuate` (page 416) reference documentation for further information.

If an instance is not suitable for live migration, it will be shut down cleanly before evacuation, respecting the *boot.host_shutdown_timeout* (page 419) configuration key.

Note

Any instance that you plan to live-migrate must have its *migration.stateful* (page 429) configuration option set to `true`. Be aware that this option can only be set while the instance is stopped. Thus, for any instance to have the ability to be live-migrated in the future, this option must be set to `true` ahead of time.

Automatic evacuation

If you set the *cluster.healing_threshold* (page 407) configuration to a non-zero value, instances are automatically evacuated if a cluster member goes offline.

When the evacuated server is available again, you must manually restore it.

Delete cluster members

To cleanly delete a member from the cluster, use the following command:

```
lxc cluster remove <member_name>
```

You can only cleanly delete members that are online and that don't have any instances located on them.

Deal with offline cluster members

If a cluster member goes permanently offline, you can force-remove it from the cluster. Make sure to do so as soon as you discover that you cannot recover the member. If you keep an offline member in your cluster, you might encounter issues when upgrading your cluster to a newer version.

To force-remove a cluster member, enter the following command on one of the cluster members that is still online:

```
lxc cluster remove --force <member_name>
```

Caution

Force-removing a cluster member will leave the member's database in an inconsistent state (for example, the storage pool on the member will not be removed). As a result, it will not be possible to re-initialize LXD later, and the server must be fully reinstalled.

Upgrade cluster members

To upgrade a cluster, you must upgrade all of its members. All members must be upgraded to the same version of LXD.

Caution

Do not attempt to upgrade your cluster if any of its members are offline. Offline members cannot be upgraded, and your cluster will end up in a blocked state.

Also note that if you are using the snap, upgrades might happen automatically, so to prevent any issues you should always recover or remove offline members immediately.

To upgrade a single member, simply upgrade the LXD package on the host and restart the LXD daemon. For example, if you are using the snap then refresh to the latest version and cohort in the current channel (also reloads LXD):

```
sudo snap refresh lxd --cohort="+"
```

If the new version of the daemon has database schema or API changes, the upgraded member might transition into a "blocked" state. In this case, the member does not serve any LXD API requests (which means that `lxc` commands don't work on that member anymore), but any running instances will continue to run.

This happens if there are other cluster members that have not been upgraded and are therefore running an older version. Run *`lxc cluster list`* (page 726) on a cluster member that is not blocked to see if any members are blocked.

As you proceed upgrading the rest of the cluster members, they will all transition to the "blocked" state. When you upgrade the last member, the blocked members will notice that all servers are now up-to-date, and the blocked members become operational again.

Update the cluster certificate

In a LXD cluster, the API on all servers responds with the same shared certificate, which is usually a standard self-signed certificate with an expiry set to ten years.

The certificate is stored at `/var/snap/lxd/common/lxd/cluster.crt` (if you use the snap) or `/var/lib/lxd/cluster.crt` (otherwise) and is the same on all cluster members.

You can replace the standard certificate with another one, such as a valid certificate obtained through ACME services (see *TLS server certificate* (page 361) for more information). To do so, run the following command on any cluster member:

```
lxc cluster update-certificate
```

This command replaces the certificate on all cluster members. For more information, see: *`lxc cluster update-certificate`* (page 733).

How to configure networks for a cluster

All members of a cluster must have identical networks defined. The only configuration keys that may differ between networks on different members are *`bridge.external_interfaces`* (page 576), *`parent`* (page 599), *`bgp.ipv4.nexthop`* (page 574), and *`bgp.ipv6.nexthop`* (page 575). See *Member configuration* (page 372) for more information.

Creating additional networks is a two-step process:

1. Define and configure the new network across all cluster members. For example, for a cluster that has three members:

```
lxc network create --target server1 my-network
lxc network create --target server2 my-network
lxc network create --target server3 my-network
```

> **Note**
>
> You can pass only the member-specific configuration keys `bridge.external_interfaces`, `parent`, `bgp.ipv4.nexthop` and `bgp.ipv6.nexthop`. Passing other configuration keys results in an error.

These commands define the network, but they don't create it. If you run *lxc network list* (page 812), you can see that the network is marked as "pending".

2. Run the following command to instantiate the network on all cluster members:

```
lxc network create my-network
```

> **Note**
>
> You can add configuration keys that are not member-specific to this command.

If you missed a cluster member when defining the network, or if a cluster member is down, you get an error.

Also see *Create a network in a cluster* (page 210).

Separate REST API and clustering networks

You can configure different networks for the REST API endpoint of your clients and for internal traffic between the members of your cluster. This separation can be useful, for example, to use a virtual address for your REST API, with DNS round robin.

To do so, you must specify different addresses for *cluster.https_address* (page 407) (the address for internal cluster traffic) and *core.https_address* (page 402) (the address for the REST API):

1. Create your cluster as usual, and make sure to use the address that you want to use for internal cluster traffic as the cluster address. This address is set as the `cluster.https_address` configuration.

2. After joining your members, set the `core.https_address` configuration to the address for the REST API. For example:

```
lxc config set core.https_address 0.0.0.0:8443
```

> **Note**
>
> `core.https_address` is specific to the cluster member, so you can use different addresses on different members. You can also use a wildcard address to make the member listen on multiple interfaces.

How to configure storage for a cluster

All members of a cluster must have identical storage pools. The only configuration keys that may differ between pools on different members are *source* (page 521), *size* (page 521), *zfs.pool_name* (page 566), *lvm.thinpool_name* (page 558) and *lvm.vg_name* (page 559). See *Member configuration* (page 372) for more information.

LXD creates a default `local` storage pool for each cluster member during initialization.

Creating additional storage pools is a two-step process:

1. Define and configure the new storage pool across all cluster members. For example, for a cluster that has three members:

   ```
   lxc storage create --target server1 data zfs source=/dev/vdb1
   lxc storage create --target server2 data zfs source=/dev/vdc1
   lxc storage create --target server3 data zfs source=/dev/vdb1 size=10GiB
   ```

   > **Note**
   >
   > You can pass only the member-specific configuration keys `source`, `size`, `zfs.pool_name`, `lvm.thinpool_name` and `lvm.vg_name`. Passing other configuration keys results in an error.

   These commands define the storage pool, but they don't create it. If you run *lxc storage list* (page 895), you can see that the pool is marked as "pending".

2. Run the following command to instantiate the storage pool on all cluster members:

   ```
   lxc storage create data zfs
   ```

   > **Note**
   >
   > You can add configuration keys that are not member-specific to this command.

   If you missed a cluster member when defining the storage pool, or if a cluster member is down, you get an error.

Also see *Create a storage pool in a cluster* (page 181).

View member-specific pool configuration

Running *lxc storage show <pool_name>* (page 896) shows the cluster-wide configuration of the storage pool.

To view the member-specific configuration, use the `--target` flag. For example:

```
lxc storage show data --target server2
```

Create storage volumes

For most storage drivers (all except for Ceph-based storage drivers), storage volumes are not replicated across the cluster and exist only on the member for which they were created. Run *lxc storage volume list <pool_name>* (page 907) to see on which member a certain volume is located.

When creating a storage volume, use the `--target` flag to create a storage volume on a specific cluster member. Without the flag, the volume is created on the cluster member on which you run the command. For example, to create a volume on the current cluster member `server1`:

```
lxc storage volume create local vol1
```

To create a volume with the same name on another cluster member:

```
lxc storage volume create local vol1 --target server2
```

Different volumes can have the same name as long as they live on different cluster members. Typical examples for this are image volumes.

You can manage storage volumes in a cluster in the same way as you do in non-clustered deployments, except that you must pass the `--target` flag to your commands if more than one cluster member has a volume with the given name. For example, to show information about the storage volumes:

```
lxc storage volume show local vol1 --target server1
lxc storage volume show local vol1 --target server2
```

How to work with a cluster:

How to manage instances in a cluster

In a cluster setup, each instance lives on one of the cluster members. You can operate each instance from any cluster member, so you do not need to log on to the cluster member on which the instance is located.

Launch an instance on a specific cluster member

When you launch an instance, you can target it to run on a specific cluster member. You can do this from any cluster member.

For example, to launch an instance named `c1` on the cluster member `server2`, use the following command:

```
lxc launch ubuntu:24.04 c1 --target server2
```

You can launch instances on specific cluster members or on specific *cluster groups* (page 292).

If you do not specify a target, the instance is assigned to a cluster member automatically. See *Automatic placement of instances* (page 372) for more information.

Check where an instance is located

To check on which member an instance is located, list all instances in the cluster:

```
lxc list
```

The location column indicates the member on which each instance is running.

Migrate an instance

You can migrate an existing instance to another cluster member. For example, to migrate the instance `c1` to the cluster member `server1`, use the following commands:

```
lxc stop c1
lxc move c1 --target server1
lxc start c1
```

See *How to migrate LXD instances between servers* (page 135) for more information.

To migrate an instance to a member of a cluster group, use the group name prefixed with `@` for the `--target` flag. For example:

```
lxc move c1 --target @group1
```

How to set up cluster groups

Cluster members can be assigned to *Cluster groups* (page 372). By default, all cluster members belong to the `default` group.

To create a cluster group, use the *`lxc cluster group create`* (page 721) command. For example:

```
lxc cluster group create gpu
```

To assign a cluster member to one or more groups, use the *`lxc cluster group assign`* (page 721) command. This command removes the specified cluster member from all the cluster groups it currently is a member of and then adds it to the specified group or groups.

For example, to assign `server1` to only the `gpu` group, use the following command:

```
lxc cluster group assign server1 gpu
```

To assign `server1` to the `gpu` group and also keep it in the `default` group, use the following command:

```
lxc cluster group assign server1 default,gpu
```

To add a cluster member to a specific group without removing it from other groups, use the *`lxc cluster group add`* (page 720) command.

For example, to add `server1` to the `gpu` group and also keep it in the `default` group, use the following command:

```
lxc cluster group add server1 gpu
```

Launch an instance on a cluster group member

With cluster groups, you can target an instance to run on one of the members of the cluster group, instead of targeting it to run on a specific member.

> **Note**
>
> *scheduler.instance* (page 602) must be set to either `all` (the default) or `group` to allow instances to be targeted to a cluster group.
>
> See *Automatic placement of instances* (page 372) for more information.

To launch an instance on a member of a cluster group, follow the instructions in *Launch an instance on a specific cluster member* (page 291), but use the group name prefixed with `@` for the `--target` flag. For example:

```
lxc launch ubuntu:24.04 c1 --target=@gpu
```

How to recover a cluster:

How to recover a cluster

It might happen that one or several members of your cluster go offline or become unreachable. If too many cluster members go offline, no operations will be possible on the cluster. See *Offline members and fault tolerance* (page 371) and *Automatic evacuation* (page 287) for more information.

If you can bring the offline cluster members back up, operation resumes as normal. If the cluster members are lost permanently (e.g. disk failure), it is possible to recover any remaining cluster members.

> **Note**
>
> When your cluster is in a state that needs recovery, most `lxc` commands do not work because the LXD database does not respond when a majority of database voters are inaccessible.
>
> The commands to recover a cluster are provided directly by the LXD daemon (`lxd`) because they modify database files directly instead of making requests to the LXD daemon.
>
> Run `lxd cluster --help` for an overview of all available commands.

Database members

Every LXD cluster has a specific number of members (configured through *cluster.max_voters* (page 408)) that serve as voting members of the distributed database. If you lose a majority of these cluster members (for example, you have a three-member cluster and you lose two members), the cluster loses quorum and becomes unavailable.

To determine which members have (or had) database roles, log on to any surviving member of your cluster and run the following command:

```
sudo lxd cluster list-database
```

Recover from quorum loss

Note

LXD automatically takes a backup of the database before making changes (see *Automated Backups* (page 296)).

If only one cluster member with the database role survives, complete the following steps. See *Reconfigure the cluster* (page 294) below for recovering more than one member.

1. Make sure that the LXD daemon is not running on the machine. For example, if you're using the snap:

   ```
   sudo snap stop lxd
   ```

2. Use the following command to reconfigure the database:

   ```
   sudo lxd cluster recover-from-quorum-loss
   ```

3. Start the LXD daemon again. For example, if you're using the snap:

   ```
   sudo snap start lxd
   ```

The database should now be back online. No information has been deleted from the database. All information about the cluster members that you have lost is still there, including the metadata about their instances. This can help you with further recovery steps if you need to re-create the lost instances.

To permanently delete the cluster members that you have lost, force-remove them. See *Delete cluster members* (page 287).

Reconfigure the cluster

Note

LXD automatically takes a backup of the database before making changes (see *Automated Backups* (page 296)).

If some members of your cluster are no longer reachable, or if the cluster itself is unreachable due to a change in IP address or listening port number, you can reconfigure the cluster.

To do so, choose the *most up-to-date database member* (page 296) to edit the cluster configuration. Once the cluster edit is complete you will need to manually copy the reconfigured global database to every other surviving member.

You can change the IP addresses or listening port numbers for each member as required. You cannot add or remove any members during this process. The cluster configuration must contain the description of the full cluster.

You can edit the *Member roles* (page 370) of the members, but with the following limitations:

- A cluster member that does not have a `database*` role cannot become a voter, because it might lack a global database.
- At least two members must remain voters (except in the case of a two-member cluster, where one voter suffices), or there will be no quorum.

Before performing the recovery, stop the LXD daemon on all surviving cluster members. For example, if you're using the snap:

```
sudo snap stop lxd
```

Complete the following steps on one database member:

1. Run the following command:

   ```
   sudo lxd cluster edit
   ```

2. Edit the YAML representation of the information that this cluster member has about the rest of the cluster:

   ```yaml
   # Latest dqlite segment ID: 1234

   members:
     - id: 1               # Internal ID of the member (Read-only)
       name: server1       # Name of the cluster member (Read-only)
       address: 192.0.2.10:8443 # Last known address of the member (Writeable)
       role: voter              # Last known role of the member (Writeable)
     - id: 2               # Internal ID of the member (Read-only)
       name: server2       # Name of the cluster member (Read-only)
       address: 192.0.2.11:8443 # Last known address of the member (Writeable)
       role: stand-by           # Last known role of the member (Writeable)
     - id: 3               # Internal ID of the member (Read-only)
       name: server3       # Name of the cluster member (Read-only)
       address: 192.0.2.12:8443 # Last known address of the member (Writeable)
       role: spare              # Last known role of the member (Writeable)
   ```

   You can edit the addresses and the roles.

3. When the cluster configuration has been changed on one member, LXD will create a tarball of the global database (`/var/snap/lxd/common/lxd/database/lxd_recovery_db.tar.gz` for snap installations or `/var/lib/lxd/database/lxd_recovery_db.tar.gz`). Copy this recovery tarball to the same path on all remaining cluster members.

   > **Note**
   >
   > The tarball can be removed from the first member after it is generated, but it does not have to be.

4. Once the tarball has been copied to all remaining cluster members, start the LXD daemon on all members again. LXD will load the recovery tarball on startup.

   If you're using the snap:

```
sudo snap start lxd
```

The cluster should now be fully available again with all surviving members reporting in. No information has been deleted from the database. All information about the cluster members and their instances is still there.

Automated Backups

LXD automatically creates a backup of the database before making changes during recovery. The backup is just a tarball of `/var/snap/lxd/common/lxd/database` (for snap users) or `/var/lib/lxd/lxd/database` (otherwise). To reset the state of the database in case of a failure, simply delete the database directory and unpack the tarball in its place:

```
cd /var/snap/lxd/common/lxd
sudo rm -r database
sudo tar -xf db_backup.TIMESTAMP.tar.gz
```

Find the most up-to-date cluster member

On every shutdown, LXD's *database members* (page 370) log the Raft term and index:

```
Dqlite last entry    index=1039 term=672
```

To determine which database member is most up to date:

- If two members have different terms, the member with the higher term is more up to date.
- If two members have the same term, the member with the higher index is more up to date.

Manually alter Raft membership

In some situations, you might need to manually alter the Raft membership configuration of the cluster because of some unexpected behavior.

For example, if you have a cluster member that was removed uncleanly, it might not show up in *`lxc cluster list`* (page 726) but still be part of the Raft configuration. To see the Raft configuration, run the following command:

```
lxd sql local "SELECT * FROM raft_nodes"
```

In that case, run the following command to remove the leftover node:

```
lxd cluster remove-raft-node <address>
```

Related topics

Explanation:

- *Clusters* (page 370)

Reference:

- *Cluster member configuration* (page 602)

2.3.2. Production setup

The following how-to guides cover common operations to prepare your LXD server setup for production.

How to check and improve the performance:

How to benchmark performance

The performance of your LXD server or cluster depends on a lot of different factors, ranging from the hardware, the server configuration, the selected storage driver and the network bandwidth to the overall usage patterns.

To find the optimal configuration, you should run benchmark tests to evaluate different setups.

LXD provides a benchmarking tool for this purpose. This tool allows you to initialize or launch a number of containers and measure the time it takes for the system to create the containers. If you run this tool repeatedly with different configurations, you can compare the performance and evaluate which is the ideal configuration.

Get the tool

To get the `lxd-benchmark` tool, you can download a pre-built binary:

1. Download the `bin.linux.lxd-benchmark` tool (`bin.linux.lxd-benchmark.aarch64`[126] or `bin.linux.lxd-benchmark.x86_64`[127]) from the **Assets** section of the latest LXD release[128].
2. Save the binary as `lxd-benchmark` and make it executable (usually by running `chmod u+x lxd-benchmark`).

If you have `go` (*Go* (page 385)) installed, you can build the tool with the following command:

```
go install github.com/canonical/lxd/lxd-benchmark@latest
```

Run the tool

Run `lxd-benchmark [action]` to measure the performance of your LXD setup.

The benchmarking tool uses the current LXD configuration, but users of the snap must export the `LXD_DIR` variable for the configuration to be found:

```
export LXD_DIR=/var/snap/lxd/common/lxd
```

If you want to use a different project, specify it with `--project`.

For all actions, you can specify the number of parallel threads to use (default is to use a dynamic batch size). You can also choose to append the results to a CSV report file and label them in a certain way.

See `lxd-benchmark help` for all available actions and flags.

[126] https://github.com/canonical/lxd/releases/latest/download/bin.linux.lxd-benchmark.aarch64
[127] https://github.com/canonical/lxd/releases/latest/download/bin.linux.lxd-benchmark.x86_64
[128] https://github.com/canonical/lxd/releases

Select an image

Before you run the benchmark, select what kind of image you want to use.

Local image

If you want to measure the time it takes to create a container and ignore the time it takes to download the image, you should copy the image to your local image store before you run the benchmarking tool.

To do so, run a command similar to the following and specify the fingerprint (for example, `2d21da400963`) of the image when you run `lxd-benchmark`:

```
lxc image copy ubuntu:24.04 local:
```

You can also assign an alias to the image and specify that alias (for example, `ubuntu`) when you run `lxd-benchmark`:

```
lxc image copy ubuntu:24.04 local: --alias ubuntu
```

Remote image

If you want to include the download time in the overall result, specify a remote image (for example, `ubuntu:24.04`). The default image that `lxd-benchmark` uses is the latest Ubuntu image (`ubuntu:`), so if you want to use this image, you can leave out the image name when running the tool.

Create and launch containers

Run the following command to create a number of containers:

```
lxd-benchmark init --count <number> <image>
```

Add `--privileged` to the command to create privileged containers.

For example:

Command	Description
`lxd-benchmark init --count 10 --privileged`	Create ten privileged containers that use the latest Ubuntu image.
`lxd-benchmark init --count 20 --parallel 4 ubuntu-minimal:24.04`	Create 20 containers that use the Ubuntu Minimal 24.04 LTS image, using four parallel threads.
`lxd-benchmark init 2d21da400963`	Create one container that uses the local image with the fingerprint `2d21da400963`.
`lxd-benchmark init --count 10 ubuntu`	Create ten containers that use the image with the alias `ubuntu`.

If you use the `init` action, the benchmarking containers are created but not started. To start the containers that you created, run the following command:

```
lxd-benchmark start
```

Alternatively, use the `launch` action to both create and start the containers:

```
lxd-benchmark launch --count 10 <image>
```

For this action, you can add the `--freeze` flag to freeze each container right after it starts. Freezing a container pauses its processes, so this flag allows you to measure the pure launch times without interference of the processes that run in each container after startup.

Delete containers

To delete the benchmarking containers that you created, run the following command:

```
lxd-benchmark delete
```

> **Note**
>
> You must delete all existing benchmarking containers before you can run a new benchmark.

How to increase the network bandwidth

You can increase the network bandwidth of your LXD setup by configuring the transmit queue length (`txqueuelen`). This change makes sense in the following scenarios:

- You have a NIC with 1 GbE or higher on a LXD host with a lot of local activity (instance-instance connections or host-instance connections).
- You have an internet connection with 1 GbE or higher on your LXD host.

The more instances you use, the more you can benefit from this tweak.

> **Note**
>
> The following instructions use a `txqueuelen` value of 10000, which is commonly used with 10GbE NICs, and a `net.core.netdev_max_backlog` value of 182757. Depending on your network, you might need to use different values.
>
> In general, you should use small `txqueuelen` values with slow devices with a high latency, and high `txqueuelen` values with devices with a low latency. For the `net.core.netdev_max_backlog` value, a good guideline is to use the minimum value of the `net.ipv4.tcp_mem` configuration.

Increase the network bandwidth on the LXD host

Ubuntu >= 18.04

Ubuntu <= 17.04

Complete the following steps to increase the network bandwidth on the LXD host:

1. Increase the transmit queue length (`txqueuelen`) of both the real NIC (for example, `enp5s0f1`) and the LXD NIC (for example, `lxdbr0`). To make the change permanent, create a file named `/etc/udev/rules.d/60-custom-txqueuelen.rules` with the following content:

```
KERNEL=="enp5s0f1", RUN+="/sbin/ip link set %k txqueuelen 10000"
KERNEL=="lxdbr0", RUN+="/sbin/ip link set %k txqueuelen 10000"
```

Apply the above `udev` rules via:

```
udevadm trigger
```

2. Increase the receive queue length (`net.core.netdev_max_backlog`). To make the change permanent, add the following configuration to `/etc/sysctl.conf`:

```
net.core.netdev_max_backlog = 182757
```

Apply the above `sysctl.conf` change via:

```
sysctl -p
```

Complete the following steps to increase the network bandwidth on the LXD host:

1. Increase the transmit queue length (`txqueuelen`) of both the real NIC and the LXD NIC (for example, `lxdbr0`). You can do this temporarily for testing with the following command:

```
ifconfig <interface> txqueuelen 10000
```

To make the change permanent, add the following command to your interface configuration in `/etc/network/interfaces`:

```
up ip link set eth0 txqueuelen 10000
```

2. Increase the receive queue length (`net.core.netdev_max_backlog`). You can do this temporarily for testing with the following command:

```
echo 182757 > /proc/sys/net/core/netdev_max_backlog
```

To make the change permanent, add the following configuration to `/etc/sysctl.conf`:

```
net.core.netdev_max_backlog = 182757
```

Increase the transmit queue length on the instances

You must also change the `txqueuelen` value for all Ethernet interfaces in your instances. To do this, use one of the following methods:

- Apply the same changes as described above for the LXD host.
- Set the `queue.tx.length` device option on the instance profile or configuration. For example, to do this for the LXD default profile:

```
lxc profile device set default eth0 queue.tx.length "10000"
```

How to monitor your server:

How to monitor metrics

LXD collects metrics for all running instances as well as some internal metrics. These metrics cover the CPU, memory, network, disk and process usage. They are meant to be consumed by Prometheus, and you can use Grafana to display the metrics as graphs. See *Provided metrics* (page 606) for lists of available metrics and *Set up a Grafana dashboard* (page 309) for instructions on how to display the metrics in Grafana.

In a cluster environment, LXD returns only the values for instances running on the server that is being accessed. Therefore, you must scrape each cluster member separately.

The instance metrics are updated when calling the `/1.0/metrics` endpoint. To handle multiple scrapers, they are cached for 8 seconds. Fetching metrics is a relatively expensive operation for LXD to perform, so if the impact is too high, consider scraping at a higher than default interval.

Query the raw data

To view the raw data that LXD collects, use the *`lxc query`* (page 869) command to query the `/1.0/metrics` endpoint:

```
~$ lxc query /1.0/metrics # HELP lxd_api_requests_completed_total The total
number of completed API requests.# TYPE lxd_api_requests_completed_total
counterlxd_api_requests_completed_total{entity_type="server",
result="error_client"}
0lxd_api_requests_completed_total{entity_type="server",result="succeeded"}
9lxd_api_requests_completed_total{entity_type="server",result="error_server"}
0lxd_api_requests_completed_total{entity_type="network",
result="error_server"}
0lxd_api_requests_completed_total{entity_type="network",
result="error_client"}
0lxd_api_requests_completed_total{entity_type="network",result="succeeded"}
0lxd_api_requests_completed_total{entity_type="cluster_member",
result="error_server"}
0lxd_api_requests_completed_total{entity_type="cluster_member",
result="error_client"}
0lxd_api_requests_completed_total{entity_type="cluster_member",
result="succeeded"}
0lxd_api_requests_completed_total{entity_type="project",result="succeeded"}
0lxd_api_requests_completed_total{entity_type="project",
result="error_server"}
0lxd_api_requests_completed_total{entity_type="project",
result="error_client"}
0lxd_api_requests_completed_total{entity_type="image",result="error_server"}
0lxd_api_requests_completed_total{entity_type="image",result="error_client"}
0lxd_api_requests_completed_total{entity_type="image",result="succeeded"}
0lxd_api_requests_completed_total{entity_type="operation",
result="error_server"}
0lxd_api_requests_completed_total{entity_type="operation",
result="error_client"}
0lxd_api_requests_completed_total{entity_type="operation",result="succeeded"}
0lxd_api_requests_completed_total{entity_type="storage_pool",
```

```
result="error_server"}
0lxd_api_requests_completed_total{entity_type="storage_pool",
result="error_client"}
0lxd_api_requests_completed_total{entity_type="storage_pool",
result="succeeded"}
0lxd_api_requests_completed_total{entity_type="warning",
result="error_server"}
0lxd_api_requests_completed_total{entity_type="warning",
result="error_client"}
0lxd_api_requests_completed_total{entity_type="warning",result="succeeded"}
0lxd_api_requests_completed_total{entity_type="identity",
result="error_client"}
0lxd_api_requests_completed_total{entity_type="identity",result="succeeded"}
0lxd_api_requests_completed_total{entity_type="identity",
result="error_server"}
0lxd_api_requests_completed_total{entity_type="profile",
result="error_server"}
0lxd_api_requests_completed_total{entity_type="profile",
result="error_client"}
0lxd_api_requests_completed_total{entity_type="profile",result="succeeded"}
0lxd_api_requests_completed_total{entity_type="instance",result="succeeded"}
2lxd_api_requests_completed_total{entity_type="instance",
result="error_server"}
0lxd_api_requests_completed_total{entity_type="instance",
result="error_client"} 0# HELP lxd_api_requests_ongoing The number of API
requests currently being handled.# TYPE lxd_api_requests_ongoing
gaugelxd_api_requests_ongoing{entity_type="server"}
1lxd_api_requests_ongoing{entity_type="network"}
0lxd_api_requests_ongoing{entity_type="cluster_member"}
0lxd_api_requests_ongoing{entity_type="project"}
0lxd_api_requests_ongoing{entity_type="image"}
0lxd_api_requests_ongoing{entity_type="operation"}
0lxd_api_requests_ongoing{entity_type="storage_pool"}
0lxd_api_requests_ongoing{entity_type="warning"}
0lxd_api_requests_ongoing{entity_type="identity"}
0lxd_api_requests_ongoing{entity_type="profile"}
0lxd_api_requests_ongoing{entity_type="instance"} 0# HELP
lxd_cpu_effective_total The total number of effective CPUs.# TYPE
lxd_cpu_effective_total
gaugelxd_cpu_effective_total{name="c",project="default",type="container"} 8#
HELP lxd_cpu_seconds_total The total number of CPU time used in seconds.#
TYPE lxd_cpu_seconds_total counterlxd_cpu_seconds_total{cpu="0",
mode="system",name="c",project="default",type="container"}
1.53794lxd_cpu_seconds_total{cpu="0",mode="user",name="c",project="default",
type="container"} 2.613658# HELP lxd_disk_read_bytes_total The total number
of bytes read.# TYPE lxd_disk_read_bytes_total
counterlxd_disk_read_bytes_total{device="nvme0n1",name="c",project="default",
type="container"} 3.6151296e+07# HELP lxd_disk_reads_completed_total The
total number of completed reads.# TYPE lxd_disk_reads_completed_total
counter...
```

Set up Prometheus

To gather and store the raw metrics, you should set up Prometheus[129]. You can then configure it to scrape the metrics through the metrics API endpoint.

Expose the metrics endpoint

To expose the `/1.0/metrics` API endpoint, you must set the address on which it should be available.

To do so, you can set either the *core.metrics_address* (page 403) server configuration option or the *core.https_address* (page 402) server configuration option. The `core.metrics_address` option is intended for metrics only, while the `core.https_address` option exposes the full API. So if you want to use a different address for the metrics API than for the full API, or if you want to expose only the metrics endpoint but not the full API, you should set the `core.metrics_address` option.

For example, to expose the full API on the `8443` port, enter the following command:

```
lxc config set core.https_address ":8443"
```

To expose only the metrics API endpoint on the `8444` port, enter the following command:

```
lxc config set core.metrics_address ":8444"
```

To expose only the metrics API endpoint on a specific IP address and port, enter a command similar to the following:

```
lxc config set core.metrics_address "192.0.2.101:8444"
```

Add a metrics certificate to LXD

Authentication for the `/1.0/metrics` API endpoint is done through a metrics certificate. A metrics certificate (type `metrics`) is different from a client certificate (type `client`) in that it is meant for metrics only and doesn't work for interaction with instances or any other LXD entities.

To create a certificate, enter the following command:

```
openssl req -x509 -newkey ec -pkeyopt ec_paramgen_curve:secp384r1 -sha384 -keyout
metrics.key -nodes -out metrics.crt -days 3650 -subj "/CN=metrics.local"
```

> **Note**
>
> The command requires OpenSSL version 1.1.0 or later.

Then add this certificate to the list of trusted clients, specifying the type as `metrics`:

```
lxc config trust add metrics.crt --type=metrics
```

[129] https://prometheus.io/

If requiring TLS client authentication isn't possible in your environment, the /1.0/metrics API endpoint can be made available to unauthenticated clients. While not recommended, this might be acceptable if you have other controls in place to restrict who can reach that API endpoint. To disable the authentication on the metrics API:

```
# Disable authentication (NOT RECOMMENDED)
lxc config set core.metrics_authentication false
```

Make the metrics certificate available for Prometheus

If you run Prometheus on a different machine than your LXD server, you must copy the required certificates to the Prometheus machine:

- The metrics certificate (metrics.crt) and key (metrics.key) that you created
- The LXD server certificate (server.crt) located in /var/snap/lxd/common/lxd/ (if you are using the snap) or /var/lib/lxd/ (otherwise)

Copy these files into a tls directory that is accessible to Prometheus, for example, /var/snap/prometheus/common/tls (if you are using the snap) or /etc/prometheus/tls (otherwise). See the following example commands:

```
# Create tls directory
mkdir /var/snap/prometheus/common/tls

# Copy newly created certificate and key to tls directory
cp metrics.crt metrics.key /var/snap/prometheus/common/tls/

# Copy LXD server certificate to tls directory
cp /var/snap/lxd/common/lxd/server.crt /var/snap/prometheus/common/tls/

# Create a symbolic link pointing to tls directory that you created
# https://bugs.launchpad.net/prometheus-snap/+bug/2066910
ln -s /var/snap/prometheus/common/tls/ /var/snap/prometheus/current/tls
```

If you are not using the snap, you must also make sure that Prometheus can read these files (usually, Prometheus is run as user prometheus):

```
chown -R prometheus:prometheus /etc/prometheus/tls
```

Configure Prometheus to scrape from LXD

Finally, you must add LXD as a target to the Prometheus configuration.

To do so, edit /var/snap/prometheus/current/prometheus.yml (if you are using the snap) or /etc/prometheus/prometheus.yaml (otherwise) and add a job for LXD.

Here's what the configuration needs to look like:

```
global:
  # How frequently to scrape targets by default. The Prometheus default value is 1m.
  scrape_interval: 15s
```

(continues on next page)

(continued from previous page)

```yaml
scrape_configs:
  - job_name: lxd
    metrics_path: '/1.0/metrics'
    scheme: 'https'
    static_configs:
      - targets: ['foo.example.com:8443']
    tls_config:
      ca_file: 'tls/server.crt'
      cert_file: 'tls/metrics.crt'
      key_file: 'tls/metrics.key'
      # XXX: server_name is required if the target name
      #      is not covered by the certificate (not in the SAN list)
      server_name: 'foo'
```

Note

- By default, the Grafana Prometheus data source assumes the `scrape_interval` to be 15 seconds. If you decide to use a different `scrape_interval` value, you must change it in both the Prometheus configuration and the Grafana Prometheus data source configuration. Otherwise, the Grafana `$__rate_interval` value will be calculated incorrectly, which might cause a `no data` response in queries that use it.
- The `server_name` must be specified if the LXD server certificate does not contain the same host name as used in the `targets` list. To verify this, open `server.crt` and check the Subject Alternative Name (SAN) section.

  For example, assume that `server.crt` has the following content:

  ```
  ~$ openssl x509 -noout -text -in
  /var/snap/prometheus/common/tls/server.crt ... X509v3 Subject
  Alternative Name: DNS:foo, IP Address:127.0.0.1, IP
  Address:0:0:0:0:0:0:0:1...
  ```

  Since the Subject Alternative Name (SAN) list doesn't include the host name provided in the `targets` list (`foo.example.com`), you must override the name used for comparison using the `server_name` directive.

Here is an example of a `prometheus.yml` configuration where multiple jobs are used to scrape the metrics of multiple LXD servers:

```yaml
global:
  # How frequently to scrape targets by default. The Prometheus default value is
1m.
  scrape_interval: 15s

scrape_configs:
  # abydos, langara and orilla are part of a single cluster (called `hdc` here)
  # initially bootstrapped by abydos which is why all 3 targets
```

(continues on next page)

(continued from previous page)

```yaml
# share the same `ca_file` and `server_name`. That `ca_file` corresponds
# to the `/var/snap/lxd/common/lxd/cluster.crt` file found on every member of
# the LXD cluster.
#
# Note: When using a certificate restricted to multiple projects,
#       use the `project` param to only scrape a specific project or projects.
#       Otherwise, omit it to return the metrics for all the accessible
#       projects in one scrape.
#
# Note: Each member of the cluster only provides metrics for instances it runs
#       locally. This is why the `lxd-hdc` cluster lists 3 targets.
- job_name: "lxd-hdc"
  metrics_path: '/1.0/metrics'
  params:
    # If no project parameter is defined, by default, metrics for all
    # accessible projects are returned.
    project: ['jdoe']
  scheme: 'https'
  static_configs:
    - targets:
      - 'abydos.hosts.example.net:8444'
      - 'langara.hosts.example.net:8444'
      - 'orilla.hosts.example.net:8444'
  tls_config:
    ca_file: 'tls/abydos.crt'
    cert_file: 'tls/metrics.crt'
    key_file: 'tls/metrics.key'
    server_name: 'abydos'

# jupiter, mars and saturn are 3 standalone LXD servers.
# Note: only the `default` project is used on them, so it is not specified.
- job_name: "lxd-jupiter"
  metrics_path: '/1.0/metrics'
  scheme: 'https'
  static_configs:
    - targets: ['jupiter.example.com:9101']
  tls_config:
    ca_file: 'tls/jupiter.crt'
    cert_file: 'tls/metrics.crt'
    key_file: 'tls/metrics.key'
    server_name: 'jupiter'

- job_name: "lxd-mars"
  metrics_path: '/1.0/metrics'
  scheme: 'https'
  static_configs:
    - targets: ['mars.example.com:9101']
  tls_config:
    ca_file: 'tls/mars.crt'
```

(continues on next page)

```
      cert_file: 'tls/metrics.crt'
      key_file: 'tls/metrics.key'
      server_name: 'mars'

  - job_name: "lxd-saturn"
    metrics_path: '/1.0/metrics'
    scheme: 'https'
    static_configs:
      - targets: ['saturn.example.com:9101']
    tls_config:
      ca_file: 'tls/saturn.crt'
      cert_file: 'tls/metrics.crt'
      key_file: 'tls/metrics.key'
      server_name: 'saturn'
```

After editing the configuration, restart Prometheus (`snap restart prometheus` if using the snap, otherwise `systemctl restart prometheus`) to start scraping.

How to send logs to Loki

LXD publishes information about its activity in the form of events. The `lxc monitor` command allows you to view this information in your shell. There are two categories of LXD events: logs and life cycle. The `lxc monitor --type=logging --pretty` command will filter and display log type events like activity of the raft cluster, for instance, while `lxc monitor --type=lifecycle --pretty` will only display life cycle events like instances starting or stopping.

In a production environment, you might want to keep a log of these events in a dedicated system. Loki[130] is one such system, and LXD provides a configuration option to forward its event stream to Loki.

Configure LXD to send logs

See the Loki documentation for instructions on installing it:

- Install Loki[131]

Once you have a Loki server up and running, you can instruct LXD to send logs to your Loki server by setting the following option:

```
lxc config set loki.api.url=http://<loki_server_IP>:3100
```

> **Note**
>
> If Loki logs are to be viewed in the Grafana dashboard, ensure the `loki.instance` configuration key matches the name of the Prometheus job. See *Set up a Grafana dashboard* (page 309).

130 https://grafana.com/oss/loki/
131 https://grafana.com/docs/loki/latest/setup/install/

Query Loki logs

Loki logs are typically viewed/queried using Grafana but Loki provides a command line utility called LogCLI allowing to query logs from your Loki server without the need for Grafana.

See the LogCLI documentation for instructions on installing it:

- Install LogCLI[132]

With your LogCLI utility up and running, first configure it to query the server you have installed before by setting the appropriate environment variable:

```
export LOKI_ADDR=http://<loki_server_IP>:3100
```

You can then query the Loki server to validate that your LXD events are getting through. LXD events all have the `app` key set to `lxd` so you can use the following `logcli` command to see LXD logs in Loki.

```
~$ logcli query -t '{app="lxd"}' 2024-02-14T21:31:20Z {app="lxd",
instance="node3", type="logging"} level="info" Updating instance
types2024-02-14T21:31:20Z {app="lxd", instance="node3", type="logging"}
level="info" Expiring log files2024-02-14T21:31:20Z {app="lxd",
instance="node3", type="logging"} level="info" Pruning resolved
warnings2024-02-14T21:31:20Z {app="lxd", instance="node3", type="logging"}
level="info" Updating images2024-02-14T21:31:20Z {app="lxd",
instance="node3", type="logging"} level="info" Done pruning resolved
warnings2024-02-14T21:31:20Z {app="lxd", instance="node3", type="logging"}
level="info" Done expiring log files2024-02-14T21:31:20Z {app="lxd",
instance="node3", type="logging"} level="info" Done updating images...
```

Add labels

LXD pushes log entries with a set of predefined labels like `app`, `project`, `instance` and `name`. To see all existing labels, you can use `logcli labels`. Some log entries might contain information in their message that you would like to access as if they were keys. In the example below, you might want to have `requester-username` as a key to query.

```
2024-02-15T22:52:25Z {app="lxd", instance="node3", location="node3", name="c1",
project="default", type="lifecycle"} requester-username="ubuntu" action="instance-
started" source="/1.0/instances/c1" requester-address="@" requester-protocol="unix
" instance-started
...
```

Use the following command to instruct LXD to move all occurrences of `requester-username="<user>"` into the label section:

```
lxc config set loki.labels="requester-username"
```

This will transform the above log entry into:

[132] https://grafana.com/docs/loki/latest/query/logcli/

```
2024-02-09T21:26:32Z {app="lxd", instance="node3", location="node3", name="c2",
project="default", requester_username="ubuntu", type="lifecycle"} action=
"instance-started" source="/1.0/instances/c2" requester-address="@" requester-
protocol="unix" instance-started
...
```

Note the replacement of - by _, as - cannot be used in keys. As `requested_username` is now a key, you can query Loki using it like this:

```
logcli query -t '{requester_username="ubuntu"}'
```

Set up a Grafana dashboard

To visualize the metrics and logs data, set up Grafana[133]. LXD provides a Grafana dashboard[134] that is configured to display the LXD metrics scraped by Prometheus and events sent to Loki.

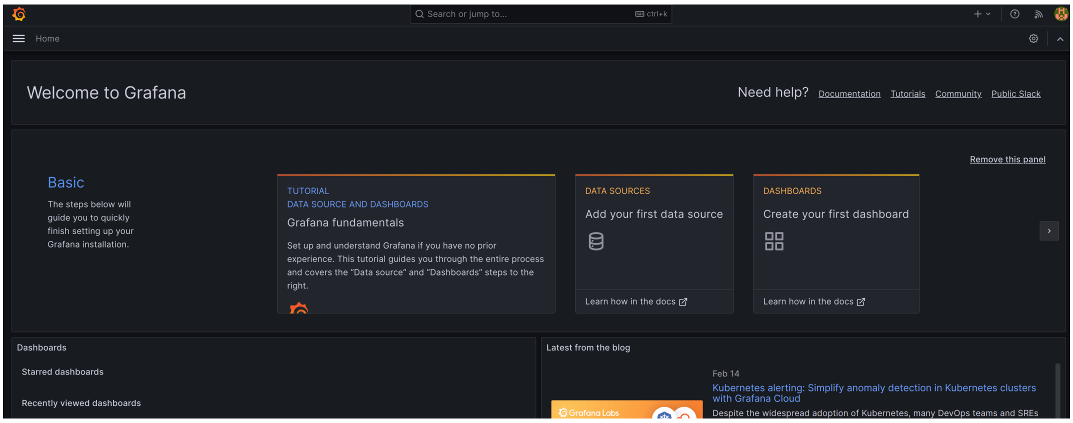
> **Note**
>
> The dashboard requires Grafana 8.4 or later.

See the Grafana documentation for instructions on installing and signing in:

- Install Grafana[135]
- Sign in to Grafana[136]

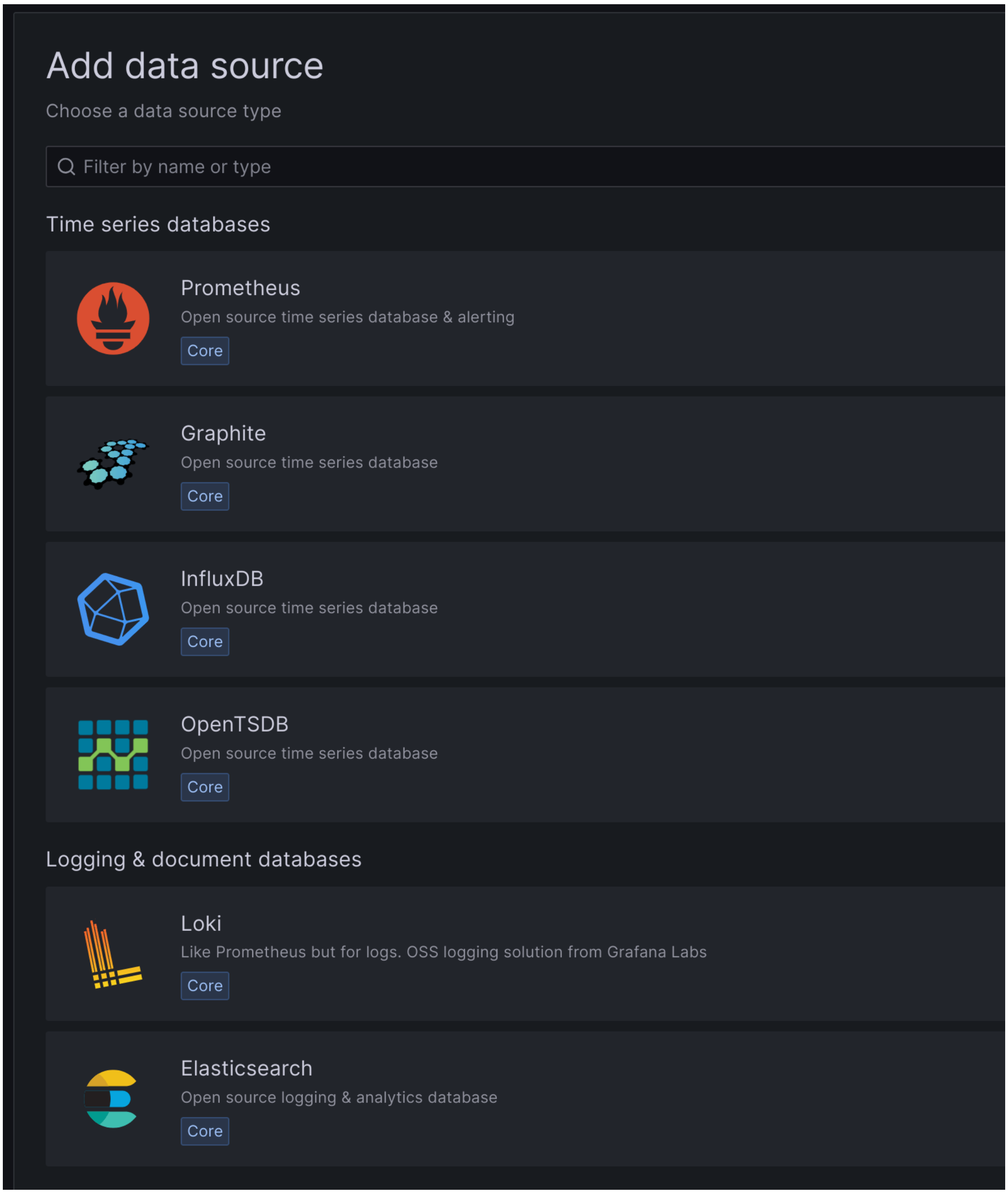
Complete the following steps to import the LXD dashboard[137]:

1. Configure Prometheus as a data source:
   1. From the Basic (quick setup) panel, choose *Data Sources*.

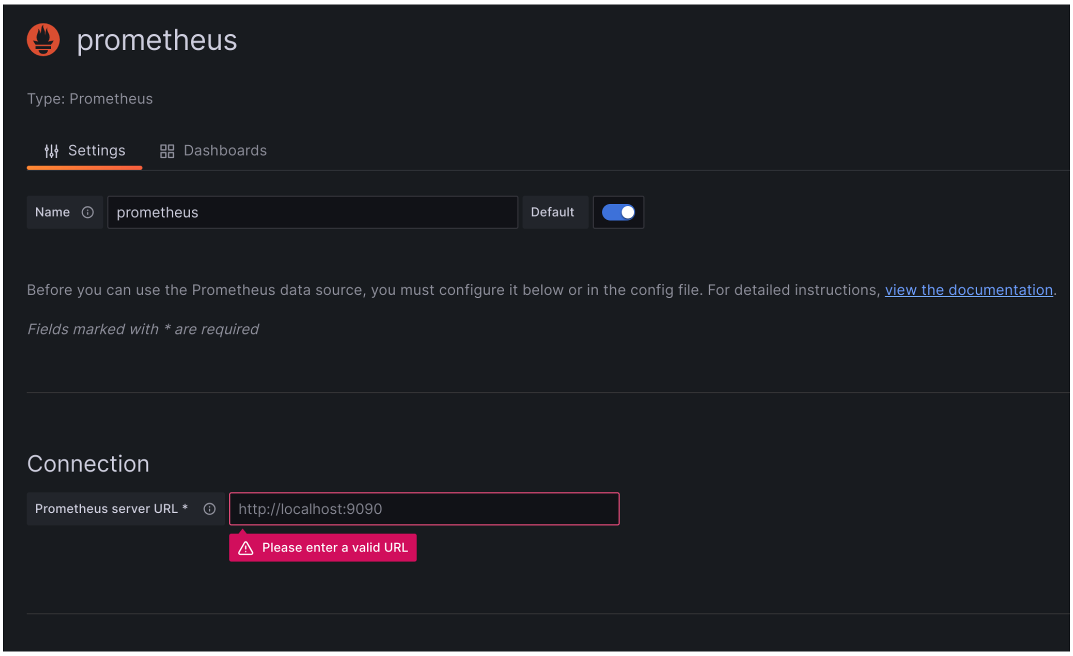


   2. Select *Prometheus*.

133 https://grafana.com/
134 https://grafana.com/grafana/dashboards/19131-lxd/
135 https://grafana.com/docs/grafana/latest/setup-grafana/installation/
136 https://grafana.com/docs/grafana/latest/setup-grafana/sign-in-to-grafana/
137 https://grafana.com/grafana/dashboards/19131-lxd/



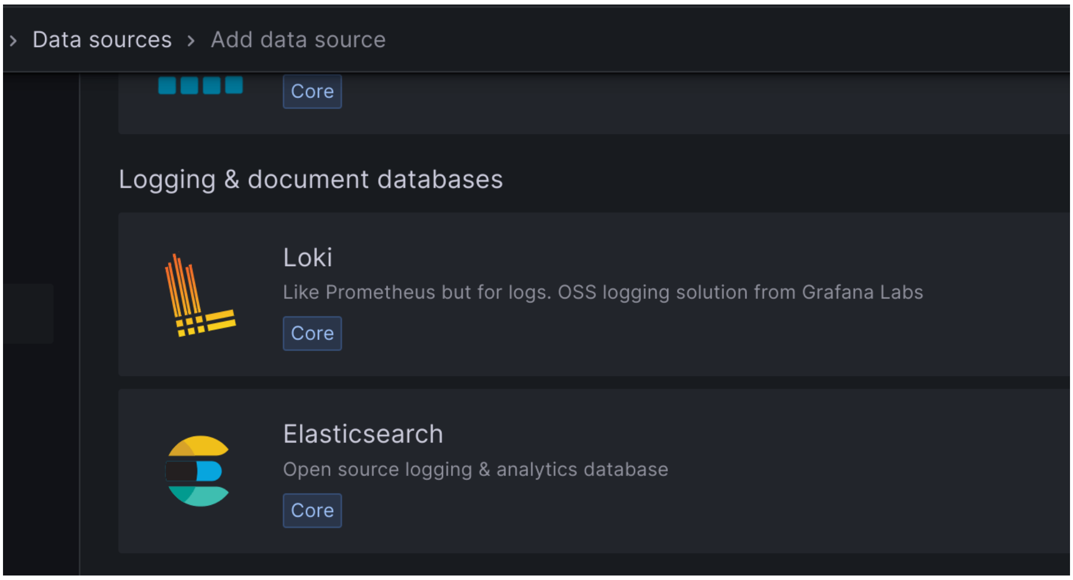
3. In the *URL* field, enter the address of your Prometheus installation (`http://localhost:9090/` if running Prometheus locally).



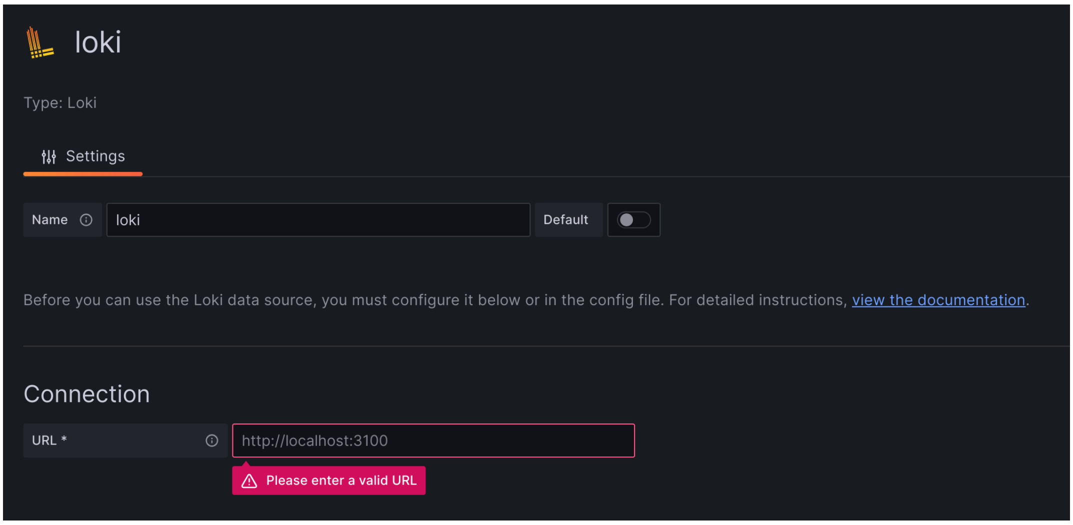
4. Keep the default configuration for the other fields and click *Save & test*.

2. Configure Loki as another data source:

1. Select *Loki*.



2. In the *URL* field, enter the address of your Loki installation (`http://localhost:3100/` if running Loki locally).

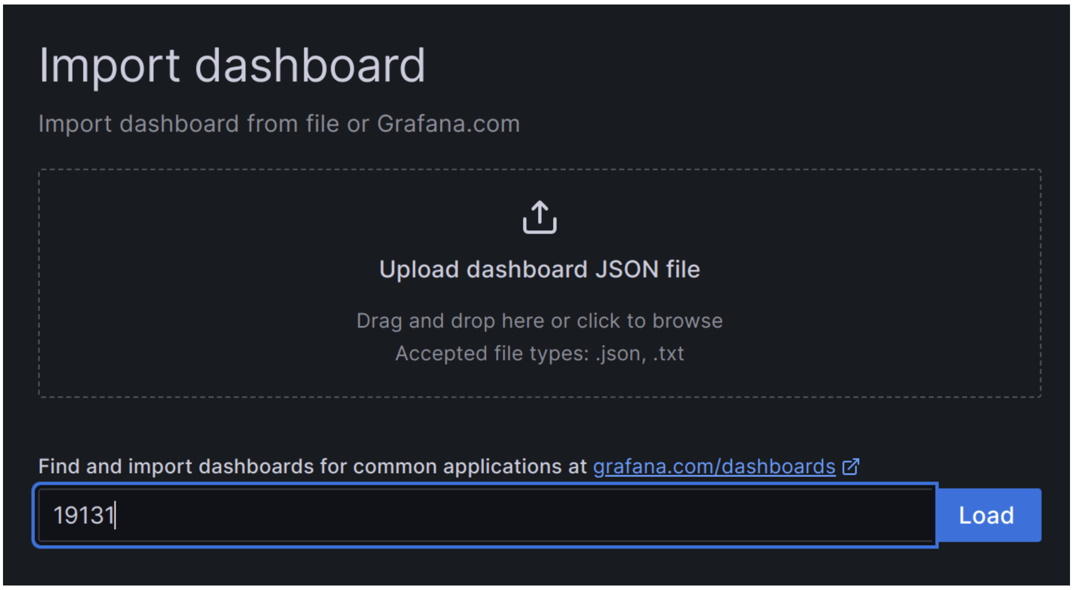


   3. Keep the default configuration for the other fields and click *Save & test*.

3. Import the LXD dashboard:

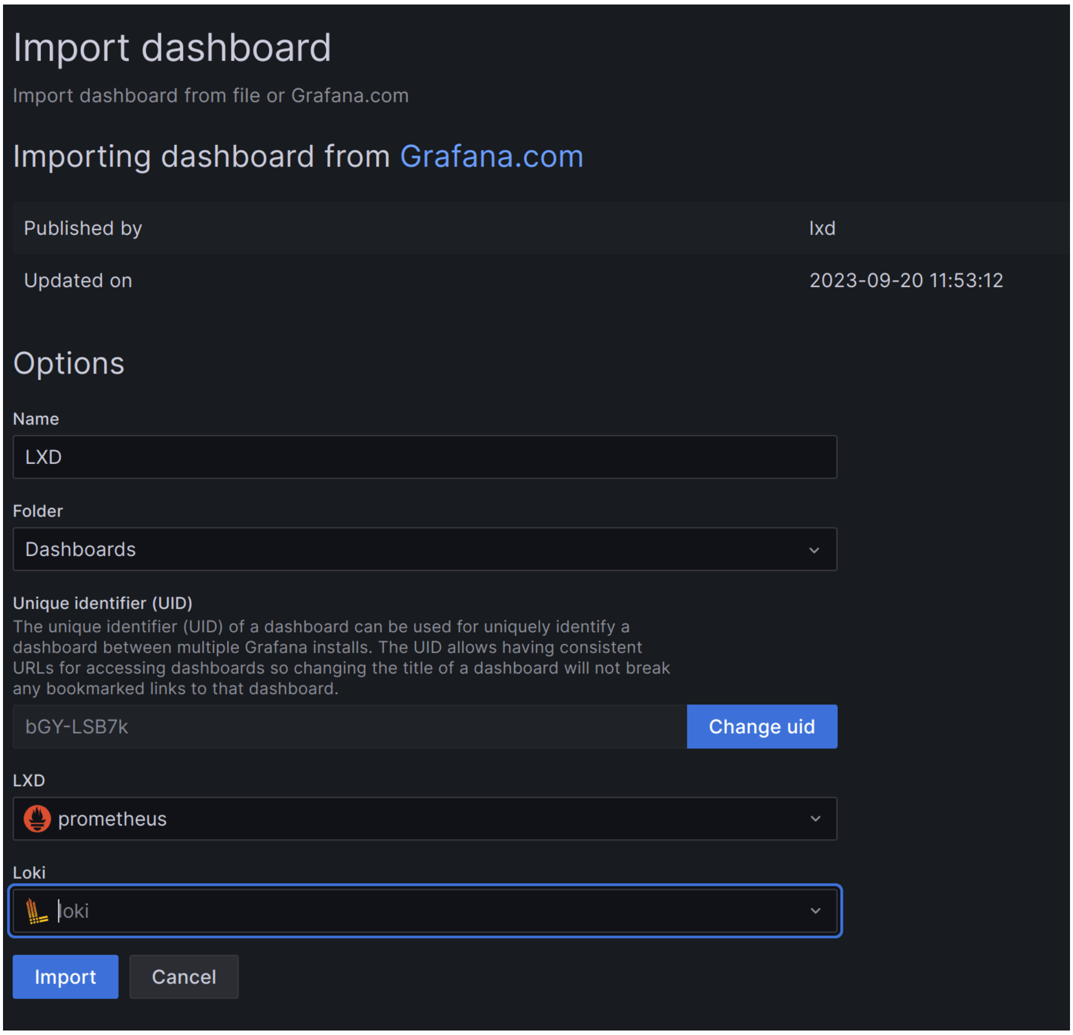
   1. Go back to the Basic (quick setup) panel and now choose *Dashboards* > *Import a dashboard*.

   2. In the *Find and import dashboards* field, enter the dashboard ID `19131`.



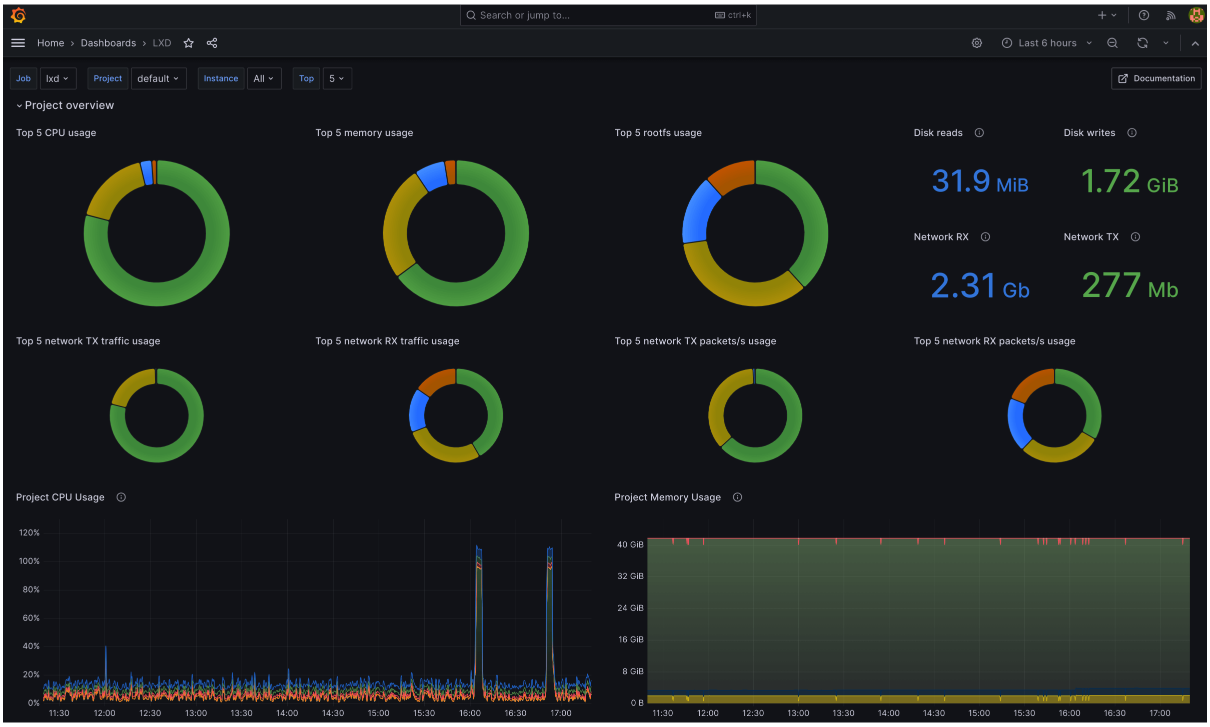
   3. Click *Load*.

   4. In the *LXD* drop-down menu, select the Prometheus and Loki data sources that you configured.

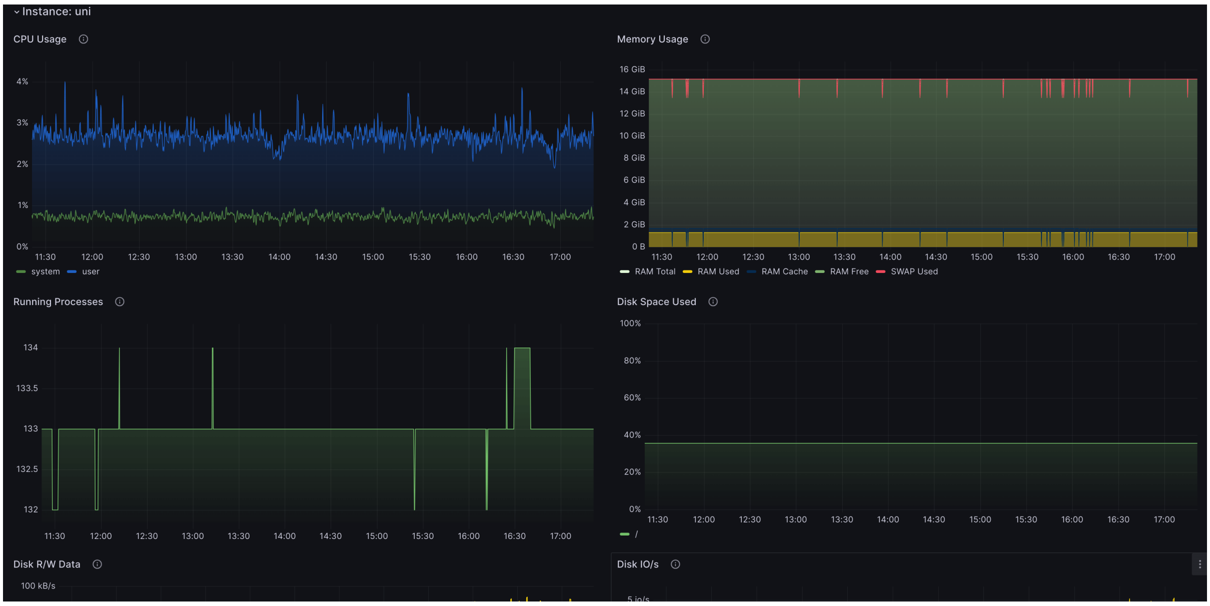


5. Click *Import*.

You should now see the LXD dashboard. You can select the project and filter by instances.



At the bottom of the page, you can see data for each instance.



Note

For proper operation of the Loki part of the dashboard, you need to ensure that the `instance` field matches the Prometheus job name. You can change the `instance` field through the *`loki.instance`* (page 410) configuration key.

The Prometheus `job_name` value can be found in `/var/snap/prometheus/current/prometheus.yml` (if you are using the snap) or `/etc/prometheus/prometheus.yaml` (otherwise).

To set the `loki.instance` configuration key, run the following command: `lxc config set loki.instance=<job_name_value>`

You can check that setting via: `lxc config get loki.instance`

Scripted setup and LXD UI integration

As an alternative to the manual steps above, we provide a script to set up the Grafana dashboard. This only supports a single-node LXD installation.

1. Launch a new instance on your LXD server:

```
lxc launch ubuntu:24.04 grafana --project default
```

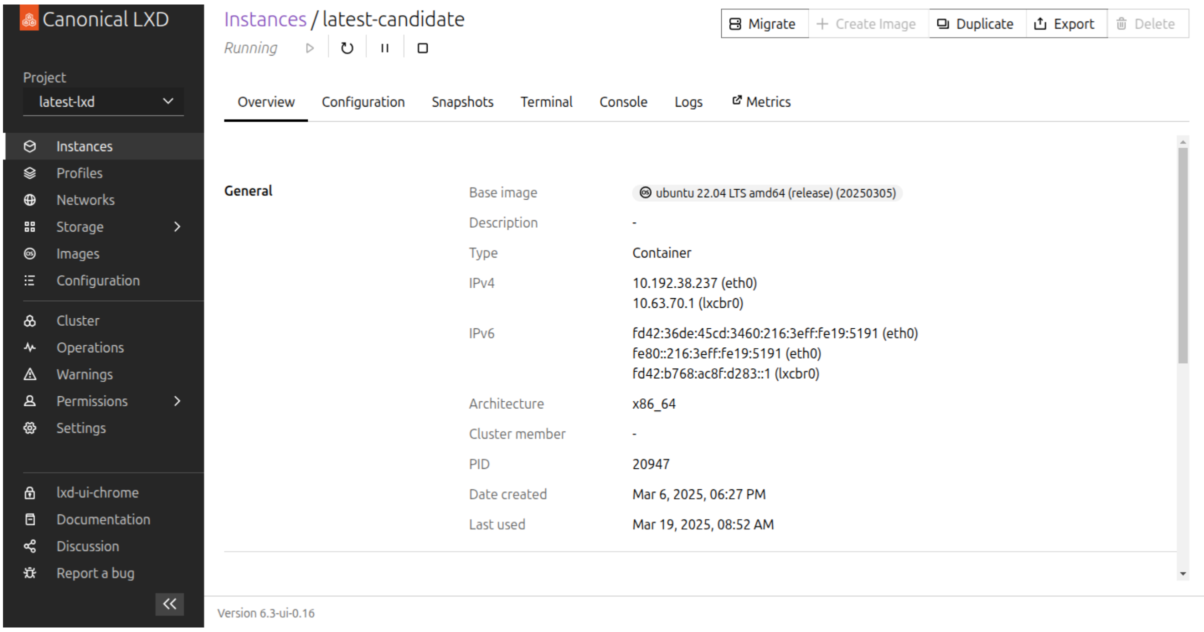
2. Run the following commands to download and execute the script to set up Grafana on the `grafana` instance:

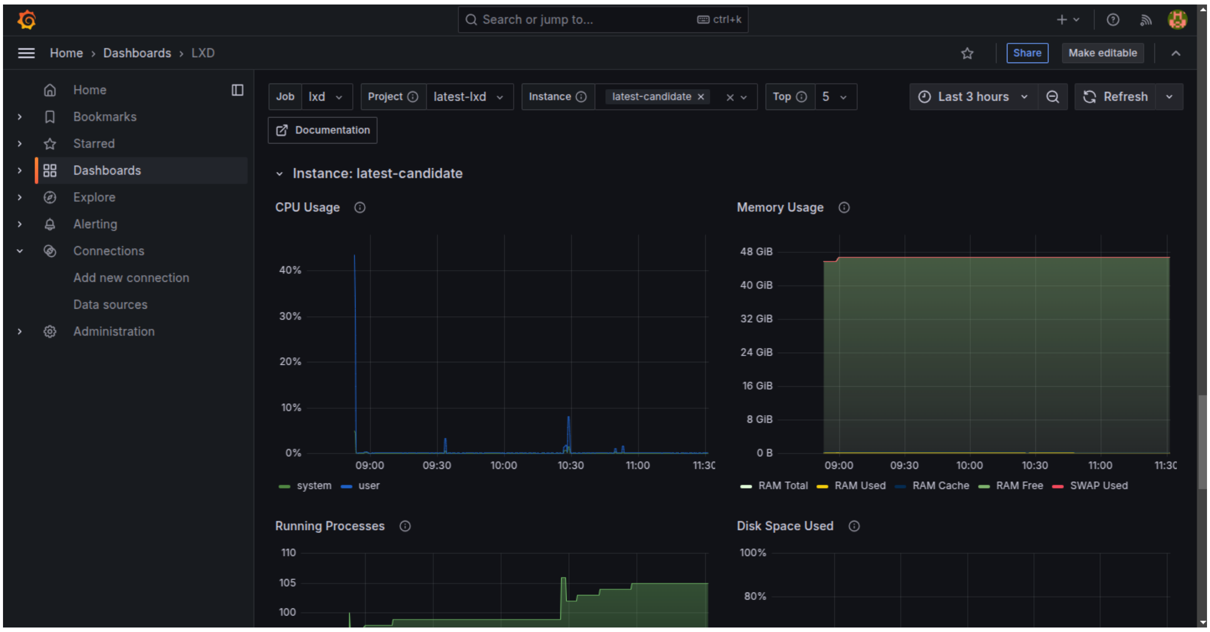
```
curl -s https://raw.githubusercontent.com/canonical/lxd/refs/heads/main/
scripts/setup-grafana.sh -o /tmp/setup-grafana.sh
chmod +x /tmp/setup-grafana.sh
/tmp/setup-grafana.sh grafana default
```

3. After the script finishes, sign in to Grafana with the default credentials `admin`/`admin` and change the password.

4. Import the LXD dashboard as described in step 3 of the manual steps in the preceding section.

The script installs Grafana, Prometheus, and Loki on a LXD instance. It also configures LXD to send metrics to Prometheus and logs to Loki. Additionally, it configures the LXD UI to be aware of the Grafana dashboard. This enables the UI to render a deep link *Metrics* to the Grafana dashboard from instance details pages (available since LXD 6.3):





How to back up your server and recover from failure:

How to back up a LXD server

In a production setup, you should always back up the contents of your LXD server.

The LXD server contains a variety of different entities, and when choosing your backup strategy, you must decide which of these entities you want to back up and how frequently you want to save them.

What to back up

The various contents of your LXD server are located on your file system and, in addition, recorded in the *LXD database* (page 356). Therefore, only backing up the database or only backing up the files on disk does not give you a full functional backup.

Your LXD server contains the following entities:

- Instances (database records and file systems)
- Images (database records, image files, and file systems)
- Networks (database records and state files)
- Profiles (database records)
- Storage volumes (database records and file systems)

Consider which of these you need to back up. For example, if you don't use custom images, you don't need to back up your images since they are available on the image server. If you use only the `default` profile, or only the standard `lxdbr0` network bridge, you might not need to worry about backing them up, because they can easily be re-created.

Full backup

To create a full backup of all contents of your LXD server, back up the `/var/snap/lxd/common/lxd` (for snap users) or `/var/lib/lxd` (otherwise) directory.

This directory contains your local storage, the LXD database, and your configuration. It does not contain separate storage devices, however. That means that whether the directory also contains the data of your instances depends on the storage drivers that you use.

Important

If your LXD server uses any external storage (for example, LVM volume groups, ZFS zpools, or any other resource that isn't directly self-contained to LXD), you must back this up separately.

See *How to back up custom storage volumes* (page 200) for instructions.

To back up your data, create a tarball of `/var/snap/lxd/common/lxd` (for snap users) or `/var/lib/lxd` (otherwise). If you are not using the snap package and your source system has a `/etc/subuid` and `/etc/subgid` file, you should also back up these files. Restoring them avoids needless shifting of instance file systems.

To restore your data, complete the following steps:

1. Stop LXD on your server (for example, with `sudo snap stop lxd`).
2. Delete the directory (`/var/snap/lxd/common/lxd` for snap users or `/var/lib/lxd` otherwise).
3. Restore the directory from the backup.
4. Delete and restore any external storage devices.
5. If you are not using the snap, restore the `/etc/subuid` and `/etc/subgid` files.
6. Restart LXD (for example, with `sudo snap start lxd` or by restarting your machine).

Export a snapshot

If you are using the LXD snap, you can also create a full backup by exporting a snapshot of the snap:

1. Create a snapshot:

   ```
   sudo snap save lxd
   ```

   Note down the ID of the snapshot (shown in the `Set` column).

2. Export the snapshot to a file:

   ```
   sudo snap export-snapshot <ID> <output_file>
   ```

See Snapshots[138] in the Snapcraft documentation for details.

[138] https://snapcraft.io/docs/snapshots

Partial backup

If you decide to only back up specific entities, you have different options for how to do this. You should consider doing some of these partial backups even if you are doing full backups in addition. It can be easier and safer to, for example, restore a single instance or reconfigure a profile than to restore the full LXD server.

Back up instances and volumes

Instances and storage volumes are backed up in a very similar way (because when backing up an instance, you basically back up its instance volume, see *Storage volume types* (page 352)).

See *How to back up instances* (page 127) and *How to back up custom storage volumes* (page 200) for detailed information. The following sections give a brief summary of the options you have for backing up instances and volumes.

Secondary backup LXD server

LXD supports copying and moving instances and storage volumes between two hosts. See *How to migrate LXD instances between servers* (page 135) and *How to move or copy storage volumes* (page 205) for instructions.

So if you have a spare server, you can regularly copy your instances and storage volumes to that secondary server to back them up. Use the `--refresh` flag to update the copies (see *Optimized volume transfer* (page 572) for the benefits).

If needed, you can either switch over to the secondary server or copy your instances or storage volumes back from it.

If you use the secondary server as a pure storage server, it doesn't need to be as powerful as your main LXD server.

Export tarballs

You can use the `export` command to export instances and volumes to a backup tarball. By default, those tarballs include all snapshots.

You can use an optimized export option, which is usually quicker and results in a smaller size of the tarball. However, you must then use the same storage driver when restoring the backup tarball.

See *Use export files for instance backup* (page 132) and *Use export files for volume backup* (page 203) for instructions.

Snapshots

Snapshots save the state of an instance or volume at a specific point in time. However, they are stored in the same storage pool and are therefore likely to be lost if the original data is deleted or lost. This means that while snapshots are very quick and easy to create and restore, they don't constitute a secure backup.

See *Use snapshots for instance backup* (page 128) and *Use snapshots for volume backup* (page 200) for more information.

Back up the database

While there is no trivial method to restore the contents of the *LXD database* (page 356), it can still be very convenient to keep a backup of its content. Such a backup can make it much easier to re-create, for example, networks or profiles if the need arises.

Use the following command to dump the content of the local database to a file:

```
lxd sql local .dump > <output_file>
```

Use the following command to dump the content of the global database to a file:

```
lxd sql global .dump > <output_file>
```

You should include these two commands in your regular LXD backup.

How to recover instances in case of disaster

LXD provides a tool for disaster recovery in case the *LXD database* (page 356) is corrupted or otherwise lost.

The tool scans the storage pools for instances and imports the instances that it finds back into the database. You need to re-create the required entities that are missing (usually profiles, projects, and networks).

Important

This tool should be used for disaster recovery only. Do not rely on this tool as an alternative to proper backups; you will lose data like profiles, network definitions, or server configuration.

The tool must be run interactively and cannot be used in automated scripts.

The tool is available through the `lxd recover` command (note the `lxd` command rather than the normal `lxc` command).

Recovery process

When you run the tool, it scans all storage pools that still exist in the database, looking for missing volumes that can be recovered. You can also specify the details of any unknown storage pools (those that exist on disk but do not exist in the database), and the tool attempts to scan those too.

After mounting the specified storage pools (if not already mounted), the tool scans them for unknown volumes that look like they are associated with LXD. LXD maintains a `backup.yaml` file in each instance's storage volume, which contains all necessary information to recover a given instance (including instance configuration, attached devices, storage volume, and pool configuration). This data can be used to rebuild the instance, storage volume, and storage pool database records. Before recovering an instance, the tool performs some consistency checks to compare what is in the `backup.yaml` file with what is actually on disk (such as matching snapshots). If all checks out, the database records are re-created.

If the storage pool database record also needs to be created, the tool uses the information from an instance's `backup.yaml` file as the basis of its configuration, rather than what the user provided during the discovery phase. However, if this information is not available, the tool falls back to restoring the pool's database record with what was provided by the user.

The tool asks you to re-create missing entities like networks. However, the tool does not know how the instance was configured. That means that if some configuration was specified through the `default` profile, you must also re-add the required configuration to the profile. For example, if the `lxdbr0` bridge is used in an instance and you are prompted to re-create it, you must add it back to the `default` profile so that the recovered instance uses it.

Example

This is how a recovery process could look:

```
~$ lxd recover This LXD server currently has the following storage pools:Would
you like to recover another storage pool? (yes/no) [default=no]: yesName of
the storage pool: defaultName of the storage backend (btrfs, ceph, cephfs,
cephobject, dir, lvm, zfs): zfsSource of the storage pool (block device,
volume group, dataset, path, ... as applicable):
/var/snap/lxd/common/lxd/storage-pools/default/containersAdditional storage
pool configuration property (KEY=VALUE, empty when done):
zfs.pool_name=defaultAdditional storage pool configuration property
(KEY=VALUE, empty when done):Would you like to recover another storage pool?
(yes/no) [default=no]:The recovery process will be scanning the following
storage pools: - NEW: "default" (backend="zfs",
source="/var/snap/lxd/common/lxd/storage-pools/default/containers")Would you
like to continue with scanning for lost volumes? (yes/no) [default=yes]:
yesScanning for unknown volumes...The following unknown volumes have been
found: - Container "u1" on pool "default" in project "default" (includes 0
snapshots) - Container "u2" on pool "default" in project "default" (includes
0 snapshots)You are currently missing the following: - Network "lxdbr0" in
project "default"Please create those missing entries and then hit ENTER:
^Z[1]+ Stopped lxd recover ~$ lxc network create lxdbr0 Network lxdbr0 created
~$ fg lxd recover The following unknown volumes have been found: - Container
"u1" on pool "default" in project "default" (includes 0 snapshots) - Container
"u2" on pool "default" in project "default" (includes 0 snapshots)Would you
like those to be recovered? (yes/no) [default=no]: yesStarting recovery... ~$
lxc list +------+---------+------+------+-----------+-----------+| NAME |
STATE | IPV4 | IPV6 | TYPE | SNAPSHOTS
|+------+---------+------+------+-----------+-----------+| u1 | STOPPED | | |
CONTAINER | 0 |+------+---------+------+------+-----------+-----------+| u2 |
STOPPED | | | CONTAINER | 0
|+------+---------+------+------+-----------+-----------+ ~$ lxc profile
device add default eth0 nic
network=lxdbr0 name=eth0 Device eth0 added to default ~$ lxc start u1 ~$ lxc list
+------+---------+-------------------+---------------------------------------
NAME | STATE | IPV4 | IPV6 | TYPE | SNAPSHOTS
|+------+---------+-------------------+--------------------------------------
u1 | RUNNING |
192.0.2.49 (eth0) | 2001:db8:8b6:abfe:216:3eff:fe82:918e (eth0) | CONTAINER | 0
|+------+---------+-------------------+--------------------------------------
u2 | STOPPED | | | CONTAINER | 0
```

```
|+------+---------+-----------------+-------------------------------------
```

Related topics

Explanation:

- *Performance tuning* (page 375)

Reference:

- *Provided metrics* (page 606)
- *Server settings for a LXD production setup* (page 602)

2.4. Miscellaneous

2.4.1. How to manage the LXD snap

The recommended way to manage LXD is its snap package[139].

For the installation guide, see: *Install the LXD snap package* (page 28). For details about the LXD snap, including its *channels* (page 389), *tracks* (page 389), and *release processes* (page 388), see: *Releases and snap* (page 388).

View snap information

To view information about the LXD snap, including the available channels and installed version, run:

```
snap info lxd
```

To view information about the installed version only, run:

```
snap list lxd
```

Sample output:

```
root@instance:~# snap list lxd Name Version Rev Tracking Publisher Noteslxd
5.21.3-c5ae129 33110 5.21/stable canonical✓ -
```

The first part of the version string corresponds to the LXD release (in this sample, `5.21.3`).

Manage updates

When LXD is *installed as a snap* (page 28), it begins tracking the specified snap channel, or the most recent stable LTS track if not specified. Whenever a new version is published to that channel, the LXD version on your system automatically updates.

For control over the update schedule, use either of the following approaches:

- *Schedule updates with the refresh timer* (page 322).
- *Hold updates* (page 322) and perform *Manual updates* (page 322) as needed.

[139] https://snapcraft.io/lxd

For clustered LXD installations, also follow the instructions below to *synchronize updates for cluster members* (page 323).

For more information about snap updates in general, see the Snap documentation: Managing updates[140].

Schedule updates with the refresh timer

Set the snaps refresh timer[141] to regularly update snaps at specific times. This enables you to schedule automatic updates during times that don't disturb normal operation. The refresh timer is set system-wide; you cannot set it for the LXD snap only. It does not apply to snaps that are held indefinitely.

For example, to configure your system to update snaps only between 8:00 am and 9:00 am on Mondays, set the following option:

```
sudo snap set system refresh.timer=mon,8:00-9:00
```

You can also use the refresh.hold[142] setting to hold all snap updates for up to 90 days, after which they automatically update. See Control updates with system options[143] in the snap documentation for details.

Hold updates

You can hold snap updates for the LXD snap, either indefinitely or for a specific duration. If you want to fully control updates to your LXD snap, you should set up an indefinite hold.

To indefinitely hold updates, run:

```
sudo snap refresh --hold lxd
```

Then you can perform *manual updates* (page 322) on a schedule that you control.

For detailed information about holds, including how to hold snaps for a specific duration rather than indefinitely, see: Pause or stop automatic updates[144] in the Snap documentation.

Manual updates

For an LXD snap installed as part of a cluster, see the section on *synchronizing cluster updates* (page 323) below.

Otherwise, run:

```
sudo snap refresh lxd
```

This updates your LXD snap to the latest release within its channel.

[140] https://snapcraft.io/docs/managing-updates
[141] https://snapcraft.io/docs/managing-updates#p-32248-refreshtimer
[142] https://snapcraft.io/docs/managing-updates#p-32248-refreshhold
[143] https://snapcraft.io/docs/managing-updates#heading--refresh-hold
[144] https://snapcraft.io/docs/managing-updates#p-32248-pause-or-stop-automatic-updates

Synchronize updates for a LXD cluster cohort

All *LXD cluster members* (page 370) must run the same LXD version. Even if you apply updates manually, versions can fall out of sync; see *Updates on clusters* (page 390) for details.

To ensure synchronized updates, set the `--cohort="+"` flag on all cluster members. You only need to set this flag once per LXD snap. This can occur during *installation* (page 28), or the first time you *perform a manual update* (page 322).

To set this flag during installation:

```
sudo snap install lxd --cohort="+"
```

To set this flag later, during a manual update:

```
sudo snap refresh lxd --cohort="+"
```

After you set this flag, `snap list lxd` shows `in-cohort` in the `Notes` column. Example:

```
root@instance:~# snap list lxd Name Version Rev Tracking Publisher Noteslxd
5.21.3-c5ae129 33110 5.21/stable canonical✓ in-cohort
```

Subsequent updates to this snap automatically use the `--cohort="+"` flag, even if you *change its channel* (page 324) or use automated or *scheduled* (page 322) updates. Thus, once the snap is `in-cohort`, you can omit that flag for future updates.

Workaround if the cohort flag malfunctions

If for some reason, the `--cohort="+"` flag does not work as expected, you can update using a matching revision on all cluster members manually:

```
sudo snap refresh lxd --revision=<revision_number>
```

Example:

```
sudo snap refresh lxd --revision=33110
```

Manage updates with an Enterprise Store proxy

For Snap Store Proxy users

If you previously used the Snap Store Proxy, see the migration guide[145] in the Enterprise Store documentation for instructions on transitioning to the Enterprise Store.

If you manage a large LXD cluster and require absolute control over when updates are applied, consider using the Enterprise Store[146]. This proxy application sits between your machines' snap clients and the Snap Store, giving you control over which snap revisions are available for installation.

[145] https://documentation.ubuntu.com/enterprise-store/main/how-to/migrate
[146] https://documentation.ubuntu.com/enterprise-store/main/

To get started, follow the Enterprise Store documentation to install[147] and register[148] the service. Once it's running, configure all cluster members to use the proxy; see Configure devices[149] for instructions. You can then override the revision[150] for the LXD snap to control which version is installed:

```
sudo enterprise-store override lxd <channel>=<revision>
```

Example:

```
sudo enterprise-store override lxd stable=25846
```

Configure the snap

The LXD snap has several configuration options that control the behavior of the installed LXD server. For example, you can define a LXD user group to achieve a multi-user environment for LXD. For more information, see: *Confine users to specific LXD projects via Unix socket* (page 175).

See the LXD snap page[151] for a list of available configuration options.

To set any of these options, run:

```
sudo snap set lxd <key>=<value>
```

Example:

```
sudo snap set lxd daemon.user.group=lxd-users
```

To see all configuration options that are explicitly set on the snap, run:

```
sudo snap get lxd
```

For more information about snap configuration options, visit Configure snaps[152] in the Snap documentation.

Change the snap channel

While it is possible to change the channel used at installation, proceed with caution.

You can upgrade (move to a newer *track* (page 389), such as from 5.21 to 6), as well as move to different *risk level* (page 390) with the same track. However, downgrading (moving to a channel with an older track, such as from 6 to 5.21) is neither recommended nor supported, as breaking changes can exist between major versions.

To change the channel, run:

```
sudo snap refresh lxd --channel=<target channel>
```

This command immediately updates the installed snap version.

[147] https://documentation.ubuntu.com/enterprise-store/main/how-to/install/
[148] https://documentation.ubuntu.com/enterprise-store/main/how-to/register/
[149] https://documentation.ubuntu.com/enterprise-store/main/how-to/devices/
[150] https://documentation.ubuntu.com/enterprise-store/main/how-to/overrides/
[151] https://snapcraft.io/lxd
[152] https://snapcraft.io/docs/configuration-in-snaps

Manage the LXD daemon

Installing LXD as a snap creates the LXD daemon as a snap service[153]. Use the following `snap` commands to manage this daemon.

To view the status of the daemon, run:

```
snap services lxd
```

To stop the daemon, run:

```
sudo snap stop lxd
```

Stopping the daemon also stops all running LXD instances.

To start the LXD daemon, run:

```
sudo snap start lxd
```

Starting the daemon also starts all previously running LXD instances.

To restart the daemon, run:

```
sudo snap restart lxd
```

This also stops and starts all running LXD instances. To keep the instances running as you restart the daemon, use the `--reload` flag:

```
sudo snap restart --reload lxd
```

For more information about managing snap services, visit Service management[154] in the Snap documentation.

Related topics

How-to guide:

- *Install the LXD snap package* (page 28)

Reference:

- *Releases and snap* (page 388)

2.4.2. Troubleshooting

If you run into problems when using LXD, check the following how-to guides to see if they help resolve your issue:

How to troubleshoot (some) Dqlite errors

Dqlite is the distributed database that LXD uses to store information that must be synchronized across a cluster. See *The LXD Dqlite database* (page 356) for more information.

This how-to guide describes strategies for how to respond to Dqlite-related errors.

[153] https://snapcraft.io/docs/service-management
[154] https://snapcraft.io/docs/service-management

Recognizing Dqlite-related errors

If LXD fails to start up or crashes, you should suspect a Dqlite-related error if the error message mentions keywords like `Dqlite`, `raft`, or `segment`.

A known risk factor for some of the errors covered below is a previous LXD crash caused by running out of disk space.

The Dqlite data directory

When investigating Dqlite-related errors, it's essential to look at the contents of the *Dqlite data directory* (page 357) for the affected node. This is the directory where the local instance of Dqlite stores all its data. You can find this directory at `/var/snap/lxd/common/lxd/database/global` (if you use the snap) or `/var/lib/lxd/database/global` (otherwise).

The data directory contains several types of file. The most important types are:

- Closed segments: These have filenames like `0000000000056436-0000000000056501`. The two numbers are the *start index* and *end index*. Both indices are inclusive.
- Open segments: These have filenames like `open-1`.
- Snapshot files: These have names like `snapshot-1-59392-27900`. The first number is the *snapshot index*.
- Snapshot metadata files: These have names like `snapshot-1-59392-27900.meta` and are paired with snapshot files.

Spotting anomalies

When looking at the contents of the data directory, watch for the following symptoms:

1. Closed segments whose index ranges overlap (remember that these ranges are inclusive).
2. A closed segment with end index X where the next closed segment has start index greater than X + 1.
3. A snapshot file with snapshot index X where the next closed segment has start index greater than X + 1.
4. A snapshot file whose size is less than the size of a previous (lower-numbered) snapshot.

When scanning for these symptoms, start with the most recent snapshots and closed segments (those with the highest indices) since the problem is more likely to be there.

Specific error messages

- `closed segment [...] is past last snapshot [...]`: This indicates that you have symptom 3 above (missing entries after a snapshot), possibly combined with symptom 1 (overlapping segments).
- `load closed segment [...]: entries count in preamble is zero`: This indicates that the mentioned segment is corrupt.

Interventions

> **Important**
>
> Before taking any of the actions below, back up the entire Dqlite data directory, so you don't lose data in case something goes wrong.

Here are some actions you can take in response to specific Dqlite errors. They are not guaranteed to work in any specific case.

- If you have overlapping closed segments (symptom 1), try deleting some of them to remove the overlap, without creating gaps in the sequence of indices or removing any index that was previously represented.
- If the snapshot file with the highest index is unexpectedly small (symptom 4), and there are still closed segments covering all the indices up to and including this snapshot's index, delete the snapshot and its corresponding metadata file.
- If the last (highest-numbered) closed segment is corrupt, try deleting it. (Deleting closed segments before the last one will create a gap and generally prevent Dqlite from starting.)

Get help

If the tips above don't help with your situation, you can always post on the LXD support forum. Make sure to mention Dqlite in your post and include the error message or messages you're seeing, LXD logs, and the output of the following command (if you're using the LXD snap):

```
sudo ls -lah /var/snap/lxd/common/lxd/database/global
```

Also mention any troubleshooting steps you've already taken and what you learned.

Additional instructions are available in the following guides:

How to debug LXD

For information on debugging instance issues, see *How to troubleshoot failing instances* (page 102).

Debugging `lxc` and `lxd`

Here are different ways to help troubleshooting `lxc` and `lxd` code.

`lxc --debug`

Adding `--debug` flag to any client command will give extra information about internals. If there is no useful info, it can be added with the logging call:

```
logger.Debugf("Hello: %s", "Debug")
```

`lxc monitor`

This command will monitor messages as they appear on remote server.

REST API through local socket

On server side the most easy way is to communicate with LXD through local socket. This command accesses `GET /1.0` and formats JSON into human readable form using jq[155] utility:

```
curl --unix-socket /var/lib/lxd/unix.socket lxd/1.0 | jq .
```

or for snap users:

```
curl --unix-socket /var/snap/lxd/common/lxd/unix.socket lxd/1.0 | jq .
```

See the *RESTful API* (page 618) for available API.

REST API through HTTPS

HTTPS connection to LXD (page 376) requires valid client certificate that is generated on first *`lxc remote add`* (page 872). This certificate should be passed to connection tools for authentication and encryption.

If desired, `openssl` can be used to examine the certificate (`~/.config/lxc/client.crt` or `~/snap/lxd/common/config/client.crt` for snap users):

```
openssl x509 -text -noout -in client.crt
```

Among the lines you should see:

```
Certificate purposes:
SSL client : Yes
```

With command line tools

```
wget --no-check-certificate --certificate=$HOME/.config/lxc/client.crt --private-key=$HOME/.config/lxc/client.key -qO - https://127.0.0.1:8443/1.0

# or for snap users
wget --no-check-certificate --certificate=$HOME/snap/lxd/common/config/client.crt --private-key=$HOME/snap/lxd/common/config/client.key -qO - https://127.0.0.1:8443/1.0
```

With browser

Some browser plugins provide convenient interface to create, modify and replay web requests. To authenticate against LXD server, convert `lxc` client certificate into importable format and import it into browser.

For example this produces `client.pfx` in Windows-compatible format:

[155] https://stedolan.github.io/jq/tutorial/

```
openssl pkcs12 -clcerts -inkey client.key -in client.crt -export -out client.pfx
```

After that, opening `https://127.0.0.1:8443/1.0` should work as expected.

Debug LXD using `pprof`

LXD provides a Go `pprof`[156] server when the *`core.debug_address`* (page 402) is set.

The debug server should not be exposed to an externally accessible address for production use cases. Use the following command to enable the server on the loopback interface:

```
lxc config set core.debug_address=localhost:8080
```

If the LXD server is running on your workstation, you can view a summary of available information by navigating to `http://localhost:8080/debug/pprof/`.

Debug the LXD database

The files of the global *database* (page 356) are stored under the `./database/global` subdirectory of your LXD data directory (e.g. `/var/lib/lxd/database/global` or `/var/snap/lxd/common/lxd/database/global` for snap users).

Since each member of the cluster also needs to keep some data which is specific to that member, LXD also uses a plain SQLite database (the "local" database), which you can find in `./database/local.db`.

Backups of the global database directory and of the local database file are made before upgrades, and are tagged with the `.bak` suffix. You can use those if you need to revert the state as it was before the upgrade.

Dumping the database content or schema

If you want to get a SQL text dump of the content or the schema of the databases, use the `lxd sql <local|global> [.dump|.schema]` command, which produces the equivalent output of the `.dump` or `.schema` directives of the `sqlite3` command line tool.

Running custom queries from the console

If you need to perform SQL queries (e.g. `SELECT`, `INSERT`, `UPDATE`) against the local or global database, you can use the `lxd sql` command (run `lxd sql --help` for details).

You should only need to do that in order to recover from broken updates or bugs. Please consult the LXD team first (creating a GitHub issue[157] or forum[158] post).

Running custom queries at LXD daemon startup

In case the LXD daemon fails to start after an upgrade because of SQL data migration bugs or similar problems, it's possible to recover the situation by creating `.sql` files containing queries that repair the broken update.

[156] https://pkg.go.dev/net/http/pprof
[157] https://github.com/canonical/lxd/issues/new
[158] https://discourse.ubuntu.com/c/lxd/126

To perform repairs against the local database, write a `./database/patch.local.sql` file containing the relevant queries, and similarly a `./database/patch.global.sql` for global database repairs.

Those files will be loaded very early in the daemon startup sequence and deleted if the queries were successful (if they fail, no state will change as they are run in a SQL transaction).

As above, please consult the LXD team first.

Syncing the cluster database to disk

If you want to flush the content of the cluster database to disk, use the `lxd sql global .sync` command, that will write a plain SQLite database file into `./database/global/db.bin`, which you can then inspect with the `sqlite3` command line tool.

Inspect a core dump file

In our continuous integration tests, we have configured the `core_pattern` as follows:

```
echo '|/bin/sh -c $@ -- eval exec gzip --fast > /var/crash/core-%e.%p.gz' | sudo
tee /proc/sys/kernel/core_pattern
```

Additionally, we have set the `GOTRACEBACK` environment variable to `crash`. Together, these ensure that when LXD crashes a core dump is compressed with `gzip` and placed in `/var/crash`.

To inspect a core dump file, you will need the LXD binary that was running at the time of the crash. The binary must include symbols; you can check this with the `file` utility. You will also need any C libraries that are used by LXD which must also include symbols.

You can inspect a core dump using Delve[159] (see the Go Wiki[160] for more information), but this does not support any dynamically linked C libraries. Instead, you can use GDB[161] which can inspect linked libraries and allows sourcing a file to load Golang support.

To do this, run:

```
gdb <LXD binary> <coredump file>
```

Then in the GDB REPL, run:

```
(gdb) source <GOROOT>/src/runtime/runtime-gdb.py
```

Substituting in the actual path to your `$GOROOT`. This will add Golang runtime support.

Finally, set the search path for C libraries using:

```
(gdb) set solib-search-path <path to C libraries>
```

You can now use the GDB REPL to inspect the core dump. Some useful commands are:

- `backtrace` (print stack trace).
- `info goroutines` (show goroutines).

[159] https://github.com/go-delve/delve
[160] https://go.dev/wiki/CoreDumpDebugging
[161] https://sourceware.org/gdb/

- `info threads` (show threads).
- `thread <thread_number>` (change thread).

Frequently asked questions

The following sections give answers to frequently asked questions. They explain how to resolve common issues and point you to more detailed information.

Why do my instances not have network access?

Most likely, your firewall blocks network access for your instances. See *How to configure your firewall* (page 258) for more information about the problem and how to fix it.

Another frequent reason for connectivity issues is running LXD and Docker on the same host. See *Prevent connectivity issues with LXD and Docker* (page 261) for instructions on how to fix such issues.

How to enable the LXD server for remote access?

By default, the LXD server is not accessible from the network, because it only listens on a local Unix socket.

You can enable it for remote access by following the instructions in *How to expose LXD to the network* (page 44).

When I do a `lxc remote add`, it asks for a token?

To be able to access the remote API, clients must authenticate with the LXD server.

See *Authenticate with the LXD server* (page 45) for instructions on how to authenticate using a trust token.

Why should I not run privileged containers?

A privileged container can do things that affect the entire host - for example, it can use things in `/sys` to reset the network card, which will reset it for the entire host, causing network blips. See *Container security* (page 378) for more information.

Almost everything can be run in an unprivileged container, or - in cases of things that require unusual privileges, like wanting to mount NFS file systems inside the container - you might need to use bind mounts.

Can I bind-mount my home directory in a container?

Yes, you can do this by using a *disk device* (page 478):

```
lxc config device add container-name home disk source=/home/${USER} path=/home/
ubuntu
```

For unprivileged containers, you need to make sure that the user in the container has working read/write permissions. Otherwise, all files will show up as the overflow UID/GID (`65536:65536`) and access to anything that's not world-readable will fail. Use either of the following methods to grant the required permissions:

- Pass `shift=true` to the *lxc config device add* (page 739) call. This depends on the kernel and file system supporting either idmapped mounts (see *lxc info* (page 782)).
- Add a `raw.idmap` entry (see *Idmaps for user namespace* (page 922)).
- Place recursive POSIX ACLs on your home directory.

Privileged containers do not have this issue because all UID/GID in the container are the same as outside. But that's also the cause of most of the security issues with such privileged containers.

How can I run Docker inside a LXD container?

To run Docker inside a LXD container, set the *security.nesting* (page 436) option of the container to `true`:

```
lxc config set <container> security.nesting true
```

If you plan to use the OverlayFS storage driver in Docker, you should also set the *security.syscalls.intercept.mknod* (page 439) and *security.syscalls.intercept.setxattr* (page 441) options to `true`. See *mknod / mknodat* (page 920) and *setxattr* (page 922) for more information.

Note that LXD containers cannot load kernel modules, so depending on your Docker configuration, you might need to have extra kernel modules loaded by the host. You can do so by setting a comma-separated list of kernel modules that your container needs:

```
lxc config set <container_name> linux.kernel_modules <modules>
```

In addition, creating a `/.dockerenv` file in your container can help Docker ignore some errors it's getting due to running in a nested environment.

Where does the LXD client (`lxc`) store its configuration?

The *lxc* (page 690) command stores its configuration under `~/.config/lxc`, or in `~/snap/lxd/common/config` for snap users.

Various configuration files are stored in that directory, for example:

- `client.crt`: client certificate (generated on demand)
- `client.key`: client key (generated on demand)
- `config.yml`: configuration file (info about `remotes`, `aliases`, etc.)
- `servercerts/`: directory with server certificates belonging to `remotes`

Why can I not ping my LXD instance from another host?

Many switches do not allow MAC address changes, and will either drop traffic with an incorrect MAC or disable the port totally. If you can ping a LXD instance from the host, but are not able to ping it from a different host, this could be the cause.

The way to diagnose this problem is to run a `tcpdump` on the uplink and you will see either ``ARP Who has `xx.xx.xx.xx` tell `yy.yy.yy.yy` ``, with you sending responses but them not getting acknowledged, or ICMP packets going in and out successfully, but never being received by the other host.

How can I monitor what LXD is doing?

To see detailed information about what LXD is doing and what processes it is running, use the *lxc monitor* (page 787) command.

For example, to show a human-readable output of all types of messages, enter the following command:

```
lxc monitor --pretty
```

See *lxc monitor --help* (page 787) for all options, and *How to debug LXD* (page 327) for more information.

Why does LXD stall when creating an instance?

Check if your storage pool is out of space (by running *lxc storage info <pool_name>* (page 894)). In that case, LXD cannot finish unpacking the image, and the instance that you're trying to create shows up as stopped.

To get more insight into what is happening, run *lxc monitor* (page 787) (see *How can I monitor what LXD is doing?* (page 333)), and check `sudo dmesg` for any I/O errors.

Why does starting containers suddenly fail?

If starting containers suddenly fails with a cgroup-related error message (`Failed to mount "/sys/fs/cgroup"`), this might be due to running a VPN client on the host.

This is a known issue for both Mullvad VPN[162] and Private Internet Access VPN[163], but might occur for other VPN clients as well. The problem is that the VPN client mounts the `net_cls` cgroup1 over cgroup2 (which LXD uses).

The easiest fix for this problem is to stop the VPN client and unmount the `net_cls` cgroup1 with the following command:

```
umount /sys/fs/cgroup/net_cls
```

If you need to keep the VPN client running, mount the `net_cls` cgroup1 in another location and reconfigure your VPN client accordingly. See this Discourse post[164] for instructions for Mullvad VPN.

Why does LXD not start on Ubuntu 20.04 LTS or earlier?

If you are running LXD on Ubuntu 20.04 LTS or earlier, you might be missing support for ZFS 2.1 in the kernel (see the *requirements* (page 386)).

If LXD fails to start, check the `/var/snap/lxd/common/lxd/logs/lxd.log` log file for the following error to see if the reason is missing ZFS support:

```
Error: Required tool 'zpool' is missing
```

[162] https://github.com/mullvad/mullvadvpn-app/issues/3651
[163] https://github.com/pia-foss/desktop/issues/50
[164] https://discuss.linuxcontainers.org/t/help-help-help-cgroup2-related-issue-on-ubuntu-jammy-with-mullvad-and-privateinter
14705/18

If you are on Ubuntu 20.04 LTS, you can resolve the issue by installing the HWE kernel and rebooting the nodes to provide the required kernel drivers for ZFS 2.1:

```
sudo apt-get update
sudo apt-get install linux-generic-hwe-20.04
```

If you are on earlier versions of Ubuntu, you should use a compatible LTS release of LXD.

Why does my VM stop responding when I try to pass through a GPU?

If you try to pass through a GPU with a large amount of VRAM, the VM might stop responding during boot or fail to start. This is often caused by the default MMIO (Memory-Mapped Input/Output) window size in QEMU being too small to map the GPU's memory.

To resolve this, stop the instance, then increase the available 64-bit PCI MMIO address space by setting the following values in *raw.qemu* (page 431):

```
lxc config set <vm-name> raw.qemu='
-global q35-pcihost.pci-hole64-size=2048G
-fw_cfg name=opt/ovmf/X-PciMmio64Mb,string=65536
'
```

These settings reserve sufficient 64-bit MMIO space in both the QEMU host and the guest firmware (OVMF (Open Virtual Machine Firmware)), which is required for GPUs with large BARs (Base Address Registers).

If you cannot resolve the issue on your own, see *How to get support* (page 334) for information about where to get help.

2.4.3. How to get support

Community support

You can seek support from the LXD developers as well as the wider community through the following channels.

Forum

Ask questions or engage in discussions: `https://discourse.ubuntu.com/c/lxd/`[165]

IRC

For live discussions, visit `#lxd`[166] on `irc.libera.chat`. See Getting started with IRC[167] if needed.

Documentation

Access the official documentation: `https://documentation.ubuntu.com/lxd/latest/`

[165] https://discourse.ubuntu.com/c/lxd/126
[166] https://web.libera.chat/#lxd
[167] https://discourse.ubuntu.com/t/getting-started-with-irc/37907

Bug reports and feature requests

To file a new bug or feature request, submit an issue on GitHub[168].

Other community resources

You can find additional resources on the LXD website[169], on YouTube[170], and the community-created tutorials[171].

Commercial support

LTS releases of LXD receive standard support for five years, which means they receive continuous updates. Commercial support for LXD is provided as part of Ubuntu Pro[172] (both Infra-only and full Ubuntu Pro), including for attached LXD instances running Ubuntu[173]. See the full service description[174] for details.

Managed solutions and firefighting support are also available for LXD deployments. See: Managed services[175].

Related topics

For information about supported releases, see: *Releases* (page 388).

2.4.4. How to contribute to LXD

The LXD team welcomes contributions through pull requests, issue reports, and discussions.

- Contribute to the code or documentation, report bugs, or request features in the GitHub repository[176]
- Ask questions or join discussions in the LXD forum[177].

Review the following guidelines before contributing to the project.

Code of Conduct

All contributors must adhere to the Ubuntu Code of Conduct[178].

License and copyright

All contributors must sign the Canonical contributor license agreement (CCLA)[179], which grants Canonical permission to use the contributions.

- You retain copyright ownership of your contributions (no copyright assignment).
- By default, contributions are licensed under the project's **AGPL-3.0-only** license.

[168] https://github.com/canonical/lxd/issues/new
[169] https://canonical.com/lxd
[170] https://www.youtube.com/channel/UCuP6xPt0WTeZu32CkQPpbvA
[171] https://discourse.ubuntu.com/c/lxd/tutorials/146
[172] https://ubuntu.com/pro
[173] https://documentation.ubuntu.com/lxd/latest/howto/ubuntu_pro_guest_attach/
[174] https://ubuntu.com/legal/ubuntu-pro-description
[175] https://ubuntu.com/managed
[176] https://github.com/canonical/lxd
[177] https://discourse.ubuntu.com/c/lxd/126
[178] https://ubuntu.com/community/ethos/code-of-conduct
[179] https://ubuntu.com/legal/contributors

- Exceptions:
  - Canonical may import code under AGPL-3.0-only compatible licenses, such as Apache-2.0.
  - Such code retains its original license and is marked as such in commit messages or file headers.
  - Some files and commits are licensed under Apache-2.0 rather than AGPL-3.0-only. These are indicated in their package-level COPYING file, file header, or commit message.

Pull requests

Submit pull requests on GitHub at: `https://github.com/canonical/lxd`.

All pull requests undergo review and must be approved before being merged into the main branch.

Commit structure

Use separate commits for different types of changes:

Type	Affects files	Commit message format
API extensions	`doc/api-extensions.md`, `shared/version/api.go`	`api: Add XYZ extension`
Documentation	Files in `doc/`	`doc: Update XYZ`
API structure	Files in `shared/api/`	`shared/api: Add XYZ`
Go client package	Files in `client/`	`client: Add XYZ`
CLI changes	Files in `lxc/`	`lxc/<command>: Change XYZ`
LXD daemon	Files in `lxd/`	`lxd/<package>: Add support for XYZ`
Tests	Files in `tests/`	`tests: Add test for XYZ`

Depending on complexity, large changes might be further split into smaller, logical commits. This commit structure facilitates the review process and simplifies backporting fixes to stable branches.

Developer Certificate of Origin sign-off

To ensure transparency and accountability in contributions to this project, all contributors must include a **Signed-off-by** line in their commits in accordance with DCO 1.1:

```
Developer Certificate of Origin
Version 1.1

Copyright (C) 2004, 2006 The Linux Foundation and its contributors.
660 York Street, Suite 102,
San Francisco, CA 94110 USA

```

(continues on next page)

```
Everyone is permitted to copy and distribute verbatim copies of this
license document, but changing it is not allowed.

Developer's Certificate of Origin 1.1

By making a contribution to this project, I certify that:

(a) The contribution was created in whole or in part by me and I
    have the right to submit it under the open source license
    indicated in the file; or

(b) The contribution is based upon previous work that, to the best
    of my knowledge, is covered under an appropriate open source
    license and I have the right under that license to submit that
    work with modifications, whether created in whole or in part
    by me, under the same open source license (unless I am
    permitted to submit under a different license), as indicated
    in the file; or

(c) The contribution was provided directly to me by some other
    person who certified (a), (b) or (c) and I have not modified
    it.

(d) I understand and agree that this project and the contribution
    are public and that a record of the contribution (including all
    personal information I submit with it, including my sign-off) is
    maintained indefinitely and may be redistributed consistent with
    this project or the open source license(s) involved.
```

Including a Signed-off-by line in your commits

Every commit must include a **Signed-off-by** line, even when part of a larger set of contributions. To do this, use the `-s` flag when committing:

```
git commit -s -m "Your commit message"
```

This automatically adds the following to your commit message:

```
Signed-off-by: Your Name <your.email@example.com>
```

By including this line, you acknowledge your agreement to the DCO 1.1 for that specific contribution.

- Use a valid name and email address—anonymous contributions are not accepted.
- Ensure your email matches the one associated with your GitHub account.

Commit signature verification

In addition to the sign-off requirement, contributors must also cryptographically sign their commits to verify authenticity. See: GitHub's documentation on commit signature verification[180].

Make-generated files

Some changes require regenerating certain files using Makefile commands.

After you run any of the commands below, you'll be prompted whether to commit the changes. If you respond `Y`, only the re-generated files are committed—any other staged files are ignored.

Formatting

If you modify any Go source files, format them:

```
make update-fmt
```

CLI tool string updates

If you modify CLI strings in `lxc/`, regenerate and commit translation files:

```
make i18n
```

API updates

If you modify the LXD API (`shared/api`), regenerate and commit the Swagger YAML file (`doc/rest-api.yaml`) used for API reference documentation:

```
make update-api
```

Configuration options updates

If you add or update configuration options, regenerate and commit the documentation metadata files (`lxd/metadata/configuration.json` and `doc/metadata.txt`):

```
make update-metadata
```

Development environment setup

Several pieces of software are needed in order to build and test LXD. Here is an easy way to create a virtual-machine to use as a development environment. LXD itself is needed to power that virtual-machine so install it first: *How to install LXD* (page 28).

Once LXD is installed and *initialized* (page 35), a special profile (`lxd-test`) needs to be loaded. The profile requires includes a `lxd-git` device (see *Types of disk devices* (page 479) for details) that will share LXD's git repository with the virtual-machine. Since this path is specific to your environment you need to adjust it when loading the profile:

[180] https://docs.github.com/en/authentication/managing-commit-signature-verification

```
# this needs to be run from inside the git repostory
GIT_ROOT="$(git rev-parse --show-toplevel)"
# create or edit the profile based on the provided template
lxc profile list | grep -qwF lxd-test || lxc profile create lxd-test
sed "s|@@PATH_TO_LXD_GIT@@|${GIT_ROOT}|" "${GIT_ROOT}/doc/lxd-test.yaml" | lxc
profile edit lxd-test
```

The `lxd-test` profile assigns CPU and memory limits similar to those available in free GitHub Action runners. Those can be adapted to the specifications of a more modest physical machine:

```
lxc profile set lxd-test limits.cpu=2
lxc profile set lxd-test limits.memory=4GiB
lxc profile device set lxd-test root size=8GiB
```

This profile can then be used to launch an Ubuntu Noble VM and start using it:

```
lxc launch ubuntu-minimal-daily:24.04 v1 --vm -p lxd-test
sleep 30
# this may take a while as many packages need to be installed
lxc exec v1 -- cloud-init status --wait --long
```

Then it is possible to build all the dependencies, LXD binaries and even run tests either automatically or manually:

```
# start a root shell in the VM
lxc exec v1 -- bash

# go into the git repo
cd lxd

# build deps and LXD binaries
make deps && make

# get an interactive test shell session with all the needed environment variables
to use and test LXD
make test-shell

# run the `exec` and `query` tests
./main.sh exec
./main.sh query

# or manually interact with LXD, for example:
lxc launch ubuntu:24.04 u1
lxc exec u1 -- hostname
lxc delete --force u1

# for a barebones test instance with just busybox (note: no IP automatically
configured)
./deps/import-busybox --alias testimage
lxc launch testimage c1
```

At this point you might want to learn more on *How to debug LXD* (page 327).

Contribute to the code

Follow the steps below to set up your development environment and start working on new LXD features.

Install LXD from source

To build the dependencies, follow the instructions in *Install LXD from source* (page 31).

Add your fork as a remote

After setting up your build environment, add your GitHub fork as a remote and fetch the latest updates:

```
git remote add myfork git@github.com:<your_username>/lxd.git
git remote update
```

Then switch to the main branch of your fork:

```
git switch myfork/main
```

Build LXD

Now you can build your fork of the project by running:

```
make
```

Before making changes, create a new branch on your fork:

```
git switch -c <name_of_your_new_branch>
```

Set up tracking for the new branch to make future pushes easier:

```
git push -u myfork <name_of_your_new_branch>
```

Important notes for new LXD contributors

- Persistent data is stored in the `LXD_DIR` directory, which is created by running `lxd init`.
  - By default, `LXD_DIR` is located at `/var/lib/lxd` (for non-snap installations) or `/var/snap/lxd/common/lxd` (for snap users).
  - To prevent version conflicts, consider setting a separate `LXD_DIR` for your development fork.
- Binaries compiled from your source are placed in `$(go env GOPATH)/bin` by default.
  - When testing, explicitly invoke these binaries instead of the global `lxd` you might have installed.
  - For convenience, you can create an alias in your `~/.bashrc` to call these binaries with the appropriate flags.

- If you have a `systemd` service running LXD from a previous installation, consider disabling it to prevent version conflicts with your development build.

Contribute to the documentation

We strive to make LXD as easy and straightforward to use as possible. To achieve this, our documentation aims to provide the information users need, cover all common use cases, and answer typical questions.

You can contribute to the documentation in several ways. We appreciate your help!

Ways to contribute

Document new features or improvements you contribute to the code.

- Submit documentation updates in pull requests alongside your code changes. We will review and merge them together with the code.

Clarify concepts or common questions based on your own experience.

- Submit a pull request with your documentation improvements.

Report documentation issues by opening an issue on GitHub[181].

- We will evaluate and update the documentation as needed.

Ask questions or suggest improvements in the LXD forum[182].

- We monitor discussions and update the documentation when necessary.

Join discussions in the `#lxd` channel on IRC via Libera Chat[183].

- While we cannot guarantee responses to IRC posts, we monitor the channel and use feedback to improve the documentation.

If you contribute images to `doc/images`:

- Use **SVG** or **PNG** formats.
- Optimize PNG images for smaller file size using a tool like TinyPNG[184] (web-based), OptiPNG[185] (CLI-based), or similar.

Documentation framework

LXD's documentation is built with Sphinx[186] and hosted on Read the Docs[187].

It is written in Markdown[188] with MyST[189] extensions. For syntax help and guidelines, see the

[181] https://github.com/canonical/lxd/issues
[182] https://discourse.ubuntu.com/c/lxd
[183] https://web.libera.chat/#lxd
[184] https://tinypng.com/
[185] https://optipng.sourceforge.net/
[186] https://www.sphinx-doc.org
[187] https://about.readthedocs.com/
[188] https://commonmark.org/
[189] https://myst-parser.readthedocs.io/

MyST style guide[190] and the documentation cheat sheet[191] (source[192]).

For structuring, the documentation uses the Diátaxis[193] approach.

Build the documentation

To build the documentation, run `make doc` from the root directory of the repository. This command installs the required tools and renders the output to the `doc/_build/` directory. To update the documentation for changed files only (without re-installing the tools), run `make doc-incremental`.

Before opening a pull request, make sure that the documentation builds without any warnings (warnings are treated as errors). To preview the documentation locally, run `make doc-serve` and go to `http://localhost:8000` to view the rendered documentation.

When you open a pull request, a preview of the documentation hosted by Read the Docs is built automatically. To see this, view the details for the `docs/readthedocs.com:canonical-lxd` check on the pull request. Others can also use this preview to validate your changes.

Automatic documentation checks

GitHub runs automatic checks on the documentation to verify the spelling, the validity of links, correct formatting of the Markdown files, and the use of inclusive language.

You can (and should!) run these tests locally before pushing your changes:

- Check the spelling: `make doc-spellcheck` (or `make doc-spelling` to first build the documentation and then check it)
- Check the validity of links: `make doc-linkcheck`
- Check the Markdown formatting: `make doc-lint`
- Check for inclusive language: `make doc-woke`

Document instructions (how-to guides)

LXD can be used with different clients, primarily the command-line interface (CLI), API, and UI. The documentation contains instructions for all of these, so when adding or updating how-to guides, remember to update the documentation for all clients.

Using tabs for client-specific information

When instructions differ between clients, use tabs to organize them:

`````
````{tabs}
```{group-tab} CLI
[...]
```
```{group-tab} API
`````

(continues on next page)

[190] https://canonical-documentation-with-sphinx-and-readthedocscom.readthedocs-hosted.com/style-guide-myst/
[191] https://documentation.ubuntu.com/lxd/latest/doc-cheat-sheet-myst/
[192] https://raw.githubusercontent.com/canonical/lxd/main/doc/doc-cheat-sheet-myst.md
[193] https://diataxis.fr/

````
[...]
```

```{group-tab} UI
[...]
```
````

> **Tip**
>
> You might need to increase the number of backticks (`) if there are code blocks or other directives in the tab content.

Guidelines for writing instructions

CLI instructions

- Link to the relevant `lxc` command reference. Example: `` [`lxc init`](lxc_init.md) ``
- You don't need to document all available command flags, but mention any that are especially relevant.
- Examples are very helpful, so add a few if it makes sense.

API instructions

- When possible, use *lxc query* (page 869) to demonstrate API calls. For complex calls, use `curl` or other widely available tools.
- In the request data, include all required fields but keep it minimal—there's no need to list every possible field.
- Link to the API call reference. Example: `` [`POST /1.0/instances`](swagger:/instances/instances_post) ``

UI instructions

- Use screenshots sparingly—they are difficult to keep up to date.
- When referring to labels in the UI, use the `{guilabel}` role. Example: `` To create an instance, go to the {guilabel}`Instances` section and click {guilabel}`Create instance`. ``

Document configuration options

Configuration options are documented by comments in the Go code. These comments are extracted automatically.

Adding or modifying configuration options

- Look for comments that start with `lxdmeta:generate` in the code.
- When adding or modifying a configuration option, include the corresponding documentation comment.
````

- Refer to the `lxd-metadata` README file[194] for formatting guidelines.
- When you add or modify configuration options, you must re-generate `doc/metadata.txt` and `lxd/metadata/configuration.json`. See the *Configuration options updates* (page 338) section for instructions.

Including configuration options in documentation

The documentation pulls sections from `doc/metadata.txt` to display a group of configuration options. For example, to include the core server options, use:

````
% Include content from [metadata.txt](metadata.txt)
```{include} metadata.txt
    :start-after: <!-- config group server-core start -->
    :end-before: <!-- config group server-core end -->
```
````

When to update documentation files

- If you add a new option to an existing group, no changes to the documentation files are needed, aside from *re-generating `metadata.txt`* (page 338). The option will be included automatically.
- If you define a new group, to add it to the documentation, you must add an `{include}` directive to the appropriate Markdown file in `doc/`.

[194] https://github.com/canonical/lxd/blob/main/lxd/lxd-metadata/README.md

3. Explanation

The explanatory guides in this section introduce you to the concepts used in LXD and help you understand how things fit together.

3.1. Important concepts

Before you start working with LXD, you need to be familiar with some important concepts about LXD and the instance types it provides.

3.1.1. `lxd` and `lxc`

LXD is frequently confused with LXC, and the fact that LXD provides both a `lxd` command and a `lxc` command doesn't make things easier.

LXD vs. LXC

LXD and LXC are two distinct implementations of Linux containers.

LXC[195] is a low-level user space interface for the Linux kernel containment features. It consists of tools (`lxc-*` commands), templates, and library and language bindings.

LXD[196] is a more intuitive and user-friendly tool aimed at making it easy to work with Linux containers. It is an alternative to LXC's tools and distribution template system, with the added features that come from being controllable over the network. Under the hood, LXD uses LXC to create and manage the containers.

LXD provides a superset of the features that LXC supports, and it is easier to use. Therefore, if you are unsure which of the tools to use, you should go for LXD. LXC should be seen as an alternative for experienced users that want to run Linux containers on distributions that don't support LXD.

LXD daemon

The central part of LXD is its daemon. It runs persistently in the background, manages the instances, and handles all requests. The daemon provides a REST API that you can access directly or through a client (for example, the default command-line client that comes with LXD).

See *Daemon behavior* (page 920) for more information about the LXD daemon.

`lxd` vs. `lxc`

To control LXD, you typically use two different commands: `lxd` and `lxc`.

LXD daemon

The `lxd` command controls the LXD daemon. Since the daemon is typically started automatically, you hardly ever need to use the `lxd` command. An exception is the `lxd init` subcommand that you run to *initialize LXD* (page 35).

There are also some subcommands for debugging and administrating the daemon, but they are intended for advanced users only. See `lxd --help` for an overview of all available subcommands.

[195] https://linuxcontainers.org/lxc/introduction/
[196] https://canonical.com/lxd

LXD client

The `lxc` command is a command-line client for LXD, which you can use to interact with the LXD daemon. You use the `lxc` command to manage your instances, the server settings, and overall the entities you create in LXD. See *`lxc --help`* (page 690) for an overview of all available subcommands.

The `lxc` tool is not the only client you can use to interact with the LXD daemon. You can also use the API, the UI, or a custom LXD client.

3.1.2. Containers and VMs

LXD provides support for two different types of *instances* (page 347): *system containers* and *virtual machines*.

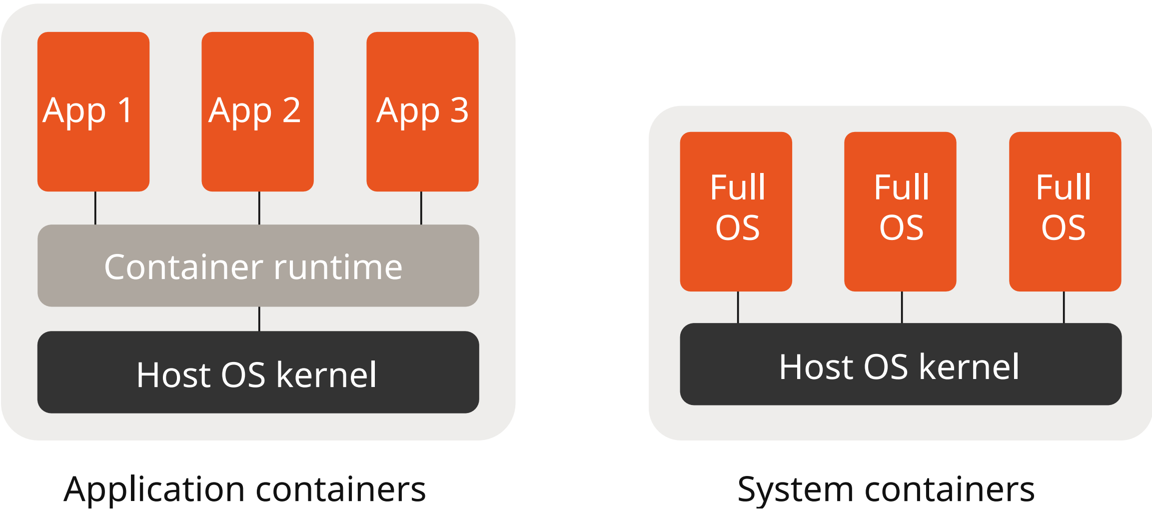
When running a system container, LXD simulates a virtual version of a full operating system. To do this, it uses the functionality provided by the kernel running on the host system.

When running a virtual machine, LXD uses the hardware of the host system, but the kernel is provided by the virtual machine. Therefore, virtual machines can be used to run, for example, a different operating system.

Application containers vs. system containers

Application containers (as provided by, for example, Docker) package a single process or application. System containers, on the other hand, simulate a full operating system and let you run multiple processes at the same time.

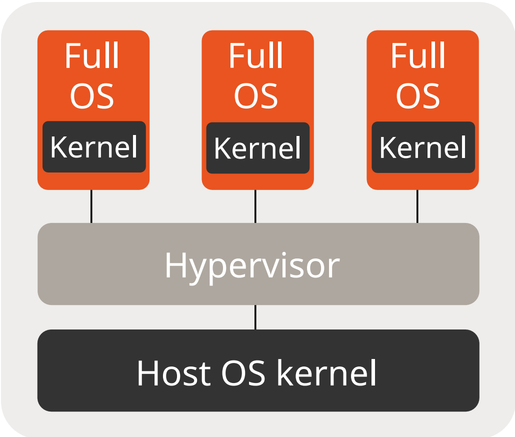
Therefore, application containers are suitable to provide separate components, while system containers provide a full solution of libraries, applications, databases, and so on. In addition, you can use system containers to create different user spaces and isolate all processes belonging to each user space, which is not what application containers are intended for.



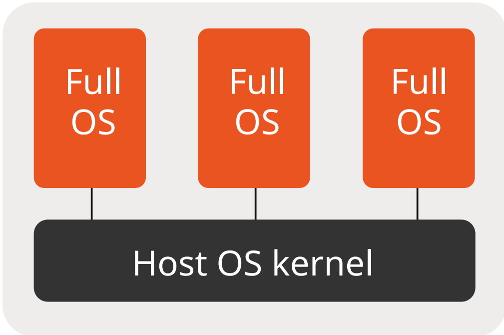
Virtual machines vs. system containers

Virtual machines emulate a physical machine, using the hardware of the host system from a full and completely isolated operating system. System containers, on the other hand, use the OS kernel of the host system instead of creating their own environment. If you run several system containers, they all share the same kernel, which makes them faster and more lightweight than virtual machines.

With LXD, you can create both system containers and virtual machines. You should use a system container to leverage the smaller size and increased performance if all functionality you require is compatible with the kernel of your host operating system. If you need functionality that is not supported by the OS kernel of your host system or you want to run a completely different OS, use a virtual machine.



Virtual machines



System containers

Instance types in LXD

LXD supports the following types of instances:

Containers

Containers are the default type for instances. They are implemented through the use of `liblxc` (LXC).

Virtual machines

Virtual machines (VMs) are natively supported since version 4.0 of LXD. Thanks to a built-in agent, they can be used almost like containers, with a similar set of features.

LXD uses `qemu` to provide the VM functionality.

> **Note**
>
> In the *Instance options* (page 415) documentation, some instance options display a `condition` field in their details, with the value of either `container` or `virtual machine`. This indicates the type of instance for which that option is available. If no `condition` field exists in an option's details, that option applies to both types.

Related topics

How-to guides:

- *Instances* (page 73)

Reference:

- *Container runtime environment* (page 398)
- *Instance configuration* (page 414)

3.2. Entities in LXD

When working with LXD, you should have a basic understanding of the different entities that are used in LXD. See the *How-to guides* (page 28) for instructions on how to work with these entities, and the following guides to understand the concepts behind them.

3.2.1. Local and remote images

LXD uses an image-based workflow. Each instance is based on an image, which contains a basic operating system (for example, a Linux distribution) and some LXD-related information.

Images are available from remote image stores (see *Remote image servers* (page 391) for an overview), but you can also create your own images, either based on an existing instances or a rootfs image.

You can copy images from remote servers to your local image store, or copy local images to remote servers. You can also use a local image to create a remote instance.

Each image is identified by a fingerprint (SHA256). To make it easier to manage images, LXD allows defining one or more aliases for each image.

Caching

When you create an instance using a remote image, LXD downloads the image and caches it locally. It is stored in the local image store with the cached flag set. The image is kept locally as a private image until either:

- The image has not been used to create a new instance for the number of days set in *`images.remote_cache_expiry`* (page 409).
- The image's expiry date (one of the image properties; see *Edit image properties* (page 152) for information on how to change it) is reached.

LXD keeps track of the image usage by updating the `last_used_at` image property every time a new instance is spawned from the image.

Auto-update

LXD can automatically keep images that come from a remote server up to date.

Note

Only images that are requested through an alias can be updated. If you request an image through a fingerprint, you request an exact image version.

Whether auto-update is enabled for an image depends on how the image was downloaded:

- If the image was downloaded and cached when creating an instance, it is automatically updated if *`images.auto_update_cached`* (page 409) was set to `true` (the default) at download time.
- If the image was copied from a remote server using the *`lxc image copy`* (page 774) command, it is automatically updated only if the `--auto-update` flag was specified.

You can change this behavior for an image by *editing the `auto_update` property* (page 152).

On startup and after every *images.auto_update_interval* (page 409) (by default, every six hours), the LXD daemon checks for more recent versions of all the images in the store that are marked to be auto-updated and have a recorded source server.

When a new version of an image is found, it is downloaded into the image store. Then any aliases pointing to the old image are moved to the new one, and the old image is removed from the store.

To not delay instance creation, LXD does not check if a new version is available when creating an instance from a cached image. This means that the instance might use an older version of an image for the new instance until the image is updated at the next update interval.

Special image properties

Image properties that begin with the prefix `requirements` (for example, `requirements.XYZ`) are used by LXD to determine the compatibility of the host system and the instance that is created based on the image. If these are incompatible, LXD does not start the instance.

The following requirements are supported:

Key	Type	Default	Description
`requirements.secureboot`	string	-	If set to `false`, indicates that the image cannot boot under secure boot.
`requirements.cgroup`	string	-	If set to `v1`, indicates that the image requires the host to run cgroup v1.
`requirements.nesting`	bool	-	If set to `true`, indicates that the image cannot work without nesting enabled.

Related topics

How-to guides:

- *Images* (page 148)

Reference:

- *Image format* (page 392)
- *Remote image servers* (page 391)

3.2.2. Storage pools, volumes, and buckets

LXD stores its data in storage pools, divided into storage volumes of different content types (like images or instances). You could think of a storage pool as the disk that is used to store data, while storage volumes are different partitions on this disk that are used for specific purposes.

In addition to storage volumes, there are storage buckets, which use the Amazon S3 (Simple Storage Service)[197] protocol. Like storage volumes, storage buckets are part of a storage pool.

[197] https://docs.aws.amazon.com/AmazonS3/latest/API/Welcome.html

Storage pools

During initialization, LXD prompts you to create a first storage pool. If required, you can create additional storage pools later (see *Create a storage pool* (page 175)).

Each storage pool uses a storage driver. The following storage drivers are supported:

- *Directory - `dir`* (page 553)
- *Btrfs - `btrfs`* (page 521)
- *LVM - `lvm`* (page 557)
- *ZFS - `zfs`* (page 563)
- *Ceph RBD - `ceph`* (page 534)
- *CephFS - `cephfs`* (page 526)
- *Ceph Object - `cephobject`* (page 532)
- *Dell PowerFlex - `powerflex`* (page 541)
- *Pure Storage - `pure`* (page 548)

See the following how-to guides for additional information:

- *How to manage storage pools* (page 175)
- *How to create an instance in a specific storage pool* (page 197)

Data storage location

Where the LXD data is stored depends on the configuration and the selected storage driver. Depending on the storage driver that is used, LXD can either share the file system with its host or keep its data separate.

Storage location	Directory	Btrfs	LVM	ZFS	Ceph (all)	Dell PowerFlex	Pure Storage
Shared with the host	✓	✓	-	✓	-	-	-
Dedicated disk/partition	-	✓	✓	✓	-	-	-
Loop disk	-	✓	✓	✓	-	-	-
Remote storage	-	-	-	-	✓	✓	✓

Shared with the host

Sharing the file system with the host is usually the most space-efficient way to run LXD. In most cases, it is also the easiest to manage.

This option is supported for the `dir` driver, the `btrfs` driver (if the host is Btrfs and you point LXD to a dedicated sub-volume) and the `zfs` driver (if the host is ZFS and you point LXD to a dedicated dataset on your zpool).

Dedicated disk or partition

Having LXD use an empty partition on your main disk or a full dedicated disk keeps its storage completely independent from the host.

This option is supported for the `btrfs` driver, the `lvm` driver and the `zfs` driver.

Loop disk

LXD can create a loop file on your main drive and have the selected storage driver use that. This method is functionally similar to using a disk or partition, but it uses a large file on your main drive instead. This means that every write must go through the storage driver and your main drive's file system, which leads to decreased performance.

The loop files reside in `/var/snap/lxd/common/lxd/disks/` if you are using the snap, or in `/var/lib/lxd/disks/` otherwise.

Loop files usually cannot be shrunk. They will grow up to the configured limit, but deleting instances or images will not cause the file to shrink. You can increase their size (quota) though; see *Resize a storage pool* (page 184).

Remote storage

The `ceph`, `cephfs` and `cephobject` drivers store the data in a completely independent Ceph storage cluster that must be set up separately. The same applies to the `powerflex` and `pure` drivers.

Default storage pool

There is no concept of a default storage pool in LXD.

When you create a storage volume, you must specify the storage pool to use.

When LXD automatically creates a storage volume during instance creation, it uses the storage pool that is configured for the instance. This configuration can be set in either of the following ways:

- Directly on an instance: *lxc launch <image> <instance_name> --storage <storage_pool>* (page 784)
- Through a profile: *lxc profile device add <profile_name> root disk path=/ pool=<storage_pool>* (page 852) and *lxc launch <image> <instance_name> --profile <profile_name>* (page 784)
- Through the default profile

In a profile, the storage pool to use is defined by the pool for the root disk device:

```
root:
  type: disk
  path: /
  pool: default
```

In the default profile, this pool is set to the storage pool that was created during initialization.

Storage volumes

When you create an instance, LXD automatically creates the required storage volumes for it. You can create additional storage volumes.

See the following how-to guides for additional information:

- *How to manage storage volumes* (page 185)
- *How to move or copy storage volumes* (page 205)
- *How to back up custom storage volumes* (page 200)

Storage volume types

Storage volumes can be of the following types:

`container/virtual-machine`

LXD automatically creates one of these storage volumes when you launch an instance. It is used as the root disk for the instance and is destroyed when the instance is deleted.

The storage pool can be explicitly specified by providing the `--storage` flag to the *launch command* (page 784). If no pool or profile is specified, LXD uses the storage pool of the default profile's root disk device.

`image`

LXD automatically creates one of these storage volumes when it unpacks an image to launch one or more instances from it. You can delete it after the instance has been created. If you do not delete it manually, it is deleted automatically ten days after it was last used to launch an instance.

The image storage volume is created in the same storage pool as the instance storage volume, and only for storage pools that use a *storage driver* (page 521) that supports optimized image storage.

`custom`

You can add one or more custom storage volumes to hold data that you want to store separately from your instances. Custom storage volumes of content type `filesystem` or `iso` can be shared between instances, and they are retained until you delete them.

You can also use custom storage volumes to hold your backups or images.

You must specify the storage pool for the custom volume when you create it.

Content types

Each storage volume uses one of the following content types:

`filesystem`

This content type is used for containers and container images. It is the default content type for custom storage volumes.

Custom storage volumes of content type `filesystem` can be attached to both containers and virtual machines, and they can be shared between instances.

`block`

This content type is used for virtual machines and virtual machine images. You can create a custom storage volume of type `block` by using the `--type=block` flag.

Custom storage volumes of content type `block` can only be attached to virtual machines. By default, they can only be attached to one instance at a time, because simultaneous access can lead to data corruption. Sharing custom storage volumes of content type `block` is made possible through the usage of the `security.shared` configuration key.

`iso`

This content type is used for custom ISO volumes. A custom storage volume of type `iso` can only be created by importing an ISO file using *`lxc storage volume import`* (page 905) or by copying another volume.

Custom storage volumes of content type `iso` can only be attached to virtual machines. They can be attached to multiple machines simultaneously as they are always read-only.

Storage buckets

Storage buckets provide object storage functionality via the S3 protocol.

They can be used in a way that is similar to custom storage volumes. However, unlike storage volumes, storage buckets are not attached to an instance. Instead, applications can access a storage bucket directly using its URL.

Each storage bucket is assigned one or more access keys, which the applications must use to access it.

Storage buckets can be located on local storage (with `dir`, `btrfs`, `lvm` or `zfs` pools) or on remote storage (with `cephobject` pools).

To enable storage buckets for local storage pool drivers and allow applications to access the buckets via the S3 protocol, you must configure the *`core.storage_buckets_address`* (page 404) server setting.

See the following how-to guide for additional information:

- *How to manage storage buckets and keys* (page 193)

Related topics

How-to guides:

- *Storage* (page 175)

Reference:

- *Storage drivers* (page 521)

3.2.3. Networking setups

There are different ways to connect your instances to the Internet. The easiest method is to have LXD create a network bridge during initialization and use this bridge for all instances, but LXD supports many different and advanced setups for networking.

Network devices

To grant direct network access to an instance, you must assign it at least one network device, also called NIC. You can configure the network device in one of the following ways:

- Use the default network bridge that you set up during the LXD initialization. Check the default profile to see the default configuration:

```
lxc profile show default
```

This method is used if you do not specify a network device for your instance.

- Use an existing network interface by adding it as a network device to your instance. This network interface is outside of LXD control. Therefore, you must specify all information that LXD needs to use the network interface.

  Use a command similar to the following:

  ```
  lxc config device add <instance_name> <device_name> nic nictype=<nic_type>
  ...
  ```

  See *Type: nic* (page 449) for a list of available NIC types and their configuration properties.

  For example, you could add a pre-existing Linux bridge (`br0`) with the following command:

  ```
  lxc config device add <instance_name> eth0 nic nictype=bridged parent=br0
  ```

- *Create a managed network* (page 210) and add it as a network device to your instance. With this method, LXD has all required information about the configured network, and you can directly attach it to your instance as a device:

  ```
  lxc network attach <network_name> <instance_name> <device_name>
  ```

  See *Attach a network to an instance* (page 212) for more information.

Managed networks

Managed networks in LXD are created and configured with the `lxc network [create|edit|set]` command.

Depending on the network type, LXD either fully controls the network or just manages an external network interface.

Note that not all *NIC types* (page 449) are supported as network types. LXD can only set up some of the types as managed networks.

Fully controlled networks

Fully controlled networks create network interfaces and provide most functionality, including, for example, the ability to do IP management.

LXD supports the following network types:

Bridge network (page 573)

A network bridge creates a virtual L2 Ethernet switch that instance NICs can connect to, making it possible for them to communicate with each other and the host. LXD bridges can leverage underlying native Linux bridges and Open vSwitch.

In LXD context, the `bridge` network type creates an L2 bridge that connects the instances that use it together into a single network L2 segment. This makes it possible to pass traffic between the instances. The bridge can also provide local DHCP and DNS.

This is the default network type.

OVN network (page 587)

OVN (Open Virtual Network) is a software-defined networking system that supports virtual network abstraction. You can use it to build your own private cloud. See `www.ovn.org`[198] for more information.

In LXD context, the `ovn` network type creates a logical network. To set it up, you must install and configure the OVN tools. In addition, you must create an uplink network that provides the network connection for OVN. As the uplink network, you should use one of the external network types or a managed LXD bridge.

> **Tip**
>
> Unlike the other network types, you can create and manage an OVN network inside a *project* (page 161). This means that you can create your own OVN network as a non-admin user, even in a restricted project.

External networks

External networks use network interfaces that already exist. Therefore, LXD has limited possibility to control them, and LXD features like network ACLs, network forwards and network zones are not supported.

The main purpose for using external networks is to provide an uplink network through a parent interface. This external network specifies the presets to use when connecting instances or other networks to a parent interface.

LXD supports the following external network types:

Macvlan network (page 593)

Macvlan is a virtual LAN (Local Area Network) that you can use if you want to assign several IP addresses to the same network interface, basically splitting up the network interface into several sub-interfaces with their own IP addresses. You can then assign IP addresses based on the randomly generated MAC addresses.

In LXD context, the `macvlan` network type provides a preset configuration to use when connecting instances to a parent macvlan interface.

SR-IOV network (page 600)

SR-IOV (Single root I/O virtualization) is a hardware standard that allows a single network card port to appear as several virtual network interfaces in a virtualized environment.

In LXD context, the `sriov` network type provides a preset configuration to use when connecting instances to a parent SR-IOV interface.

Physical network (page 595)

The `physical` network type connects to an existing physical network, which can be a network interface or a bridge, and serves as an uplink network for OVN.

It provides a preset configuration to use when connecting OVN networks to a parent interface.

[198] https://www.ovn.org/

Recommendations

In general, if you can use a managed network, you should do so because networks are easy to configure and you can reuse the same network for several instances without repeating the configuration.

Which network type to choose depends on your specific use case. If you choose a fully controlled network, it provides more functionality than using a network device.

As a general recommendation:

- If you are running LXD on a single system or in a public cloud, use a *Bridge network* (page 573), possibly in connection with the Ubuntu Fan[199].
- If you are running LXD in your own private cloud, use an *OVN network* (page 587).

> **Note**
>
> OVN requires a shared L2 uplink network for proper operation. Therefore, using OVN is usually not possible if you run LXD in a public cloud.

- To connect an instance NIC to a managed network, use the `network` property rather than the `parent` property, if possible. This way, the NIC can inherit the settings from the network and you don't need to specify the `nictype`.

Related topics

How-to guides:

- *Networking* (page 210)

Reference:

- *Networks* (page 573)

3.2.4. The LXD Dqlite database

LXD uses a distributed database to store the server configuration and state, which allows for quicker queries than if the configuration was stored inside each instance's directory (as it is done by LXC, for example).

To understand the advantages, consider a query against the configuration of all instances, like "what instances are using `br0`?". To answer that question without a database, you would have to iterate through every single instance, load and parse its configuration, and then check which network devices are defined in there. With a database, you can run a simple query on the database to retrieve this information.

Dqlite

In a LXD cluster, all members of the cluster must share the same database state. Therefore, LXD uses Dqlite[200], a distributed version of SQLite. Dqlite provides replication, fault-tolerance, and automatic failover without the need of external database processes.

When using LXD as a single machine and not as a cluster, the Dqlite database effectively behaves like a regular SQLite database.

[199] https://www.youtube.com/watch?v=5cwd0vZJ5bw
[200] https://dqlite.io/

File location

The database files are stored in the `database` sub-directory of your LXD data directory (thus `/var/snap/lxd/common/lxd/database/` if you use the snap, or `/var/lib/lxd/database/` otherwise).

Upgrading LXD to a newer version might require updating the database schema. In this case, LXD automatically stores a backup of the database and then runs the update. See *Upgrade LXD* (page 35) for more information.

Backup

See *Back up the database* (page 319) for instructions on how to back up the contents of the LXD database.

3.2.5. `lxc show` and `info`

For the entities managed by LXD, the `lxc` command provides a `list` sub-command, and might provide `show` and `info` sub-commands. The purpose of the `info` sub-command is to show current state information, and the purpose of the `show` sub-command is to show configuration information and how the entity is used by other entities.

For example, the `lxc network info` command shows IP address and traffic statistics:

```
Name: lxdbr0
MAC address: 00:16:3e:d3:ec:41
MTU: 1500
State: up

Ips:
  inet    192.0.2.1
  inet6   2001:db8:f4a1:53d2::1
  inet6   fe80::216:3eff:fed3:ec41

Network usage:
  Bytes received: 127.66kB
  Bytes sent: 15.54kB
  Packets received: 1433
  Packets sent: 175
```

The `lxc network show` command, on the other hand, shows how the network is configured, and which entities are using the network:

```
config:
  ipv4.address: 192.0.2.1/24
  ipv4.nat: "true"
  ipv6.address: 2001:db8:f4a1:53d2::1/64
  ipv6.nat: "true"
description: ""
name: lxdbr0
type: bridge
used_by:
- /1.0/instances/ubuntu
- /1.0/profiles/default
```

(continues on next page)

```
managed: true
status: Created
locations:
- none
```

Refer to the manual pages for details of the commands for managing entities:

- Instances: *lxc list* (page 785), *lxc info* (page 782)
- Images: *lxc image list* (page 778), *lxc image info* (page 777), *lxc image show* (page 780)
- Networks: *lxc network list* (page 812), *lxc network info* (page 812), *lxc network show* (page 830)
- Profiles: *lxc profile list* (page 857), *lxc profile show* (page 860)
- Projects: *lxc project list* (page 865), *lxc project info* (page 865), *lxc project show* (page 867)
- Storage: *lxc storage list* (page 895), *lxc storage info* (page 894), *lxc storage show* (page 896)

3.3. Access management

In LXD, access to the API is controlled through TLS or OpenID Connect authentication. When using OpenID Connect, you can grant permissions to access specific entities to different clients. You can also restrict access to LXD entities by confining them to specific projects.

3.3.1. Remote API authentication

Remote communications with the LXD daemon happen using JSON over HTTPS. This requires the LXD API to be exposed over the network; see *How to expose LXD to the network* (page 44) for instructions.

To be able to access the remote API, clients must authenticate with the LXD server. The following authentication methods are supported:

- *TLS client certificates* (page 358)
- *OpenID Connect authentication* (page 361)

TLS client certificates

When using TLS (Transport Layer Security) client certificates for authentication, both the client and the server will generate a key pair the first time they're launched. The server will use that key pair for all HTTPS connections to the LXD socket. The client will use its certificate as a client certificate for any client-server communication.

To cause certificates to be regenerated, simply remove the old ones. On the next connection, a new certificate is generated.

Communication protocol

The supported protocol must be TLS 1.3 or better.

All communications must use perfect forward secrecy, and ciphers must be limited to strong elliptic curve ones (such as ECDHE-RSA or ECDHE-ECDSA).

Any generated key should be at least 4096 bit RSA, preferably 384 bit ECDSA. When using signatures, only SHA-2 signatures should be trusted.

Since we control both client and server, there is no reason to support any backward compatibility to broken protocol or ciphers.

Trusted TLS clients

The workflow to authenticate with the server is similar to that of SSH, where an initial connection to an unknown server triggers a prompt:

1. When the user adds a server with *`lxc remote add`* (page 872), the server is contacted over HTTPS, its certificate is downloaded and the fingerprint is shown to the user.
2. The user is asked to confirm that this is indeed the server's fingerprint, which they can manually check by connecting to the server or by asking someone with access to the server to run the info command and compare the fingerprints.
3. The server attempts to authenticate the client:
   - If the client certificate is in the server's trust store, the connection is granted.
   - If the client certificate is not in the server's trust store, the server prompts the user for a token. If the provided token matches, the client certificate is added to the server's trust store and the connection is granted. Otherwise, the connection is rejected.

See *How to expose LXD to the network* (page 44) and *Authenticate with the LXD server* (page 45) for instructions on how to configure TLS authentication and add trusted clients.

Using a PKI system

In a PKI (Public key infrastructure) setup, a system administrator manages a central PKI that issues client certificates for all the LXD clients and server certificates for all the LXD daemons.

In PKI mode, TLS authentication requires that client certificates are signed be the CA (Certificate authority). This requirement does not apply to clients that authenticate via *OIDC* (page 361).

The steps for enabling PKI mode differ slightly depending on whether you use an ACME provider in addition (see *TLS server certificate* (page 361)).

Only PKI

PKI and ACME

If you use a PKI system, both the server and client certificates are issued by intermediate CA(s). The `client.ca` file contains the certificate used by the client to verify the server certificate it receives when making a connection to a remote. The `server.ca` file contains the certificate used by the server to verify the client certificate associated with an incoming connection.

Both files contain trust anchors used to evaluate if the received leaf certificate from the other end of the connection is to be trusted or not. If the leaf certificate's chain of trust leads to one of the trusted anchors it will be trusted (unless revoked).

1. Add the CA certificate to all machines:

- Place the `client.ca` file in the clients' configuration directories (`~/.config/lxc` or `~/snap/lxd/common/config` for snap users).
- Place the `server.ca` file in the server's configuration directory (`/var/lib/lxd` or `/var/snap/lxd/common/lxd` for snap users).

> **Note**
>
> In a cluster setup, the CA certificate must be named `cluster.ca`, and the same file must be added to all cluster members.

2. Place the certificates issued by the CA in the clients' configuration directories, replacing the automatically generated `client.crt` and `client.key` files.
3. If you want clients to automatically trust the server, place the certificates issued by the CA in the server's configuration directory, replacing the automatically generated `server.crt` and `server.key` files.

   > **Note**
   >
   > In a cluster setup, the certificate files must be named `cluster.crt` and `cluster.key`. They must be identical on all cluster members.

   When a client adds a PKI-enabled server or cluster as a remote, it checks the server certificate and prompts the user to trust the server certificate only if the certificate has not been signed by the CA.
4. Restart the LXD daemon.

If you use a PKI system alongside an ACME provider, the server certificates are issued by the ACME provider, and the client certificates are issued by a secondary CA.

1. Place the CA certificate for the server (`server.ca`) in the server's configuration directory (`/var/lib/lxd` or `/var/snap/lxd/common/lxd` for snap users), so that the server can authenticate the clients.

   > **Note**
   >
   > In a cluster setup, the CA certificate must be named `cluster.ca`, and the same file must be added to all cluster members.

2. Place the certificates issued by the CA in the clients' configuration directories, replacing the automatically generated `client.crt` and `client.key` files.
3. Restart the LXD daemon.

Trusting certificates

CA-signed client certificates are not automatically trusted. You must still add them to the server in one of the ways described in *Trusted TLS clients* (page 359).

To automatically trust CA-signed client certificates, set the *`core.trust_ca_certificates`* (page 405) server configuration to true. When `core.trust_ca_certificates` is enabled, any new clients with a CA-signed certificate will have full access to LXD.

Revoking certificates

To revoke certificates via the PKI, place a certificate revocation list in the server's configuration directory as `ca.crl` and restart the LXD daemon. A client with a CA-signed certificate that has been revoked, and is present in `ca.crl`, will not be able to authenticate with LXD, nor add LXD as a remote via *mutual TLS* (page 359).

OpenID Connect authentication

LXD supports using OpenID Connect[201] to authenticate users through an OIDC (OpenID Connect) Identity Provider.

To configure LXD to use OIDC authentication, set the *`oidc.*`* (page 406) server configuration options. Your OIDC provider must be configured to enable the Device Authorization Grant[202] type.

To add a remote pointing to a LXD server configured with OIDC authentication, run *`lxc remote add <remote_name> <remote_address>`* (page 872). You are then prompted to authenticate through your web browser, where you must confirm that the device code displayed in the browser matches the device code that is displayed in the terminal window. The LXD client then retrieves and stores an access token, which it provides to LXD for all interactions. The identity provider might also provide a refresh token. In this case, the LXD client uses this refresh token to attempt to retrieve another access token when the current access token has expired.

> **Warning**
>
> Only set `oidc.client.secret` if required by the Identity Provider. Once set, this key allows the LXD UI client to authenticate. However, the secret is not shared with other LXD clients (such as the LXD CLI).

When an OIDC client initially authenticates with LXD, it does not have access to the majority of the LXD API. OIDC clients must be granted access by an administrator, see *Fine-grained authorization* (page 364).

TLS server certificate

LXD supports issuing server certificates using ACME (Automatic Certificate Management Environment) services, for example, Let's Encrypt[203].

To enable this feature, set the following server configuration:

- *`acme.domain`* (page 405): The domain for which the certificate should be issued.
- *`acme.email`* (page 405): The email address used for the account of the ACME service.
- *`acme.agree_tos`* (page 405): Must be set to `true` to agree to the ACME service's terms of service.

[201] https://openid.net/developers/how-connect-works/
[202] https://oauth.net/2/device-flow/
[203] https://letsencrypt.org/

- *acme.ca_url* (page 405): The directory URL of the ACME service. By default, LXD uses "Let's Encrypt".

For this feature to work, LXD must be reachable from port 80. This can be achieved by using a reverse proxy such as HAProxy[204].

Here's a minimal HAProxy configuration that uses `lxd.example.net` as the domain. After the certificate has been issued, LXD will be reachable from `https://lxd.example.net/`.

```
# Global configuration
global
  log /dev/log local0
  chroot /var/lib/haproxy
  stats socket /run/haproxy/admin.sock mode 660 level admin
  stats timeout 30s
  user haproxy
  group haproxy
  daemon
  ssl-default-bind-options ssl-min-ver TLSv1.2
  tune.ssl.default-dh-param 2048
  maxconn 100000

# Default settings
defaults
  mode tcp
  timeout connect 5s
  timeout client 30s
  timeout client-fin 30s
  timeout server 120s
  timeout tunnel 6h
  timeout http-request 5s
  maxconn 80000

# Default backend - Return HTTP 301 (TLS upgrade)
backend http-301
  mode http
  redirect scheme https code 301

# Default backend - Return HTTP 403
backend http-403
  mode http
  http-request deny deny_status 403

# HTTP dispatcher
frontend http-dispatcher
  bind :80
  mode http

  # Backend selection
  tcp-request inspect-delay 5s
```

(continues on next page)

[204] http://www.haproxy.org/

```
  # Dispatch
  default_backend http-403
  use_backend http-301 if { hdr(host) -i lxd.example.net }

# SNI dispatcher
frontend sni-dispatcher
  bind :443
  mode tcp

  # Backend selection
  tcp-request inspect-delay 5s

  # require TLS
  tcp-request content reject unless { req.ssl_hello_type 1 }

  # Dispatch
  default_backend http-403
  use_backend lxd-nodes if { req.ssl_sni -i lxd.example.net }

# LXD nodes
backend lxd-nodes
  mode tcp

  option tcp-check

  # Multiple servers should be listed when running a cluster
  server lxd-node01 1.2.3.4:8443 check
  server lxd-node02 1.2.3.5:8443 check
  server lxd-node03 1.2.3.6:8443 check
```

Failure scenarios

In the following scenarios, authentication is expected to fail.

Server certificate changed

The server certificate might change in the following cases:

- The server was fully reinstalled and therefore got a new certificate.
- The connection is being intercepted (MITM (Machine in the middle)).

In such cases, the client will refuse to connect to the server because the certificate fingerprint does not match the fingerprint in the configuration for this remote.

It is then up to the user to contact the server administrator to check if the certificate did in fact change. If it did, the certificate can be replaced by the new one, or the remote can be removed altogether and re-added.

Server trust relationship revoked

The server trust relationship is revoked for a client if another trusted client or the local server administrator removes the trust entry for the client on the server.

In this case, the server still uses the same certificate, but all API calls return a 403 code with an error indicating that the client isn't trusted.

Related topics

Explanation:

- *Security* (page 376)

How-to guides:

- *How to expose LXD to the network* (page 44)

3.3.2. Remote API authorization

When LXD is *exposed over the network* (page 44) it is possible to restrict API access via two mechanisms:

- *Restricted TLS certificates* (page 364)
- *Fine-grained authorization* (page 364)

Restricted TLS certificates

It is possible to restrict a *TLS client* (page 359) to one or multiple projects. In this case, the client will also be prevented from performing global configuration changes or altering the configuration (limits, restrictions) of the projects it's allowed access to.

To restrict access, use *lxc config trust edit <fingerprint>* (page 753). Set the `restricted` key to `true` and specify a list of projects to restrict the client to. If the list of projects is empty, the client will not be allowed access to any of them.

Fine-grained authorization

It is possible to restrict *OIDC clients* (page 361) and fine-grained TLS identities to granular actions on specific LXD resources. For example, one could restrict a user to be able to view, but not edit, a single instance.

There are four key concepts that LXD uses to manage these fine-grained permissions:

- **Entitlements**: An entitlement encapsulates an action that can be taken against a LXD API resource type. Some entitlements might apply to many resource types, whereas other entitlements can only apply to a single resource type. For example, the entitlement `can_view` is available for all resource types, but the entitlement `can_exec` is only available for LXD resources of type `instance`.
- **Permissions**: A permission is the application of an entitlement to a particular LXD resource. For example, given the entitlement `can_exec` that is only defined for instances, a permission is the combination of `can_exec` and a single instance, as uniquely defined by its API URL (for example, `/1.0/instances/c1?project=foo`).
- **Identities (users)**: An identity is any authenticated party that makes requests to LXD, including TLS clients. When an OIDC client adds a LXD server as a remote, the OIDC client is saved in LXD as an identity. Permissions cannot be assigned to identities directly.

- **Groups**: A group is a collection of one or more identities. Identities can belong to one or more groups. Permissions can be assigned to groups. TLS clients cannot currently be assigned to groups.

Explore permissions

To discover available permissions that can be assigned to a group, or view permissions that are currently assigned, run the following command:

```
lxc auth permission list --max-entitlements 0
```

The entity type column displays the LXD API resource type, this value is required when adding a permission to a group.

The URL column displays the URL of the LXD API resource.

The entitlements column displays all available entitlements for that entity type. If any groups are already assigned permissions on the API resource at the displayed URL, they are listed alongside the entitlements that they have been granted.

Some useful permissions at a glance:

- The `admin` entitlement on entity type `server` gives full access to LXD. This is equivalent to an unrestricted TLS client or Unix socket access.
- The `project_manager` entitlement on entity type `server` grants access to create, edit, and delete projects, and all resources belonging to those projects. However, this permission does not allow access to server configuration, storage pool configuration, or certificate/identity management.
- The `operator` entitlement on entity type `project` grants access to create, edit, and delete all resources belonging to the project against which the permission is granted. Members of a group with this permission will not be able to edit the project configuration itself. This is equivalent to a restricted TLS client with access to the same project.
- The `user` entitlement on entity type `instance` grants access to view an instance, pull/push files, get a console, and begin a terminal session. Members of a group with this entitlement cannot edit the instance configuration.

For a full list, see *Permissions* (page 609).

> **Note**
>
> Due to a limitation in the LXD client, if `can_exec` is granted to a group for a particular instance, members of the group will not be able to start a terminal session unless `can_view_events` is additionally granted for the parent project of the instance. We are working to resolve this.

Explore identities

To discover available identities that can be assigned to a group, or view identities that are currently assigned, run the following command:

```
lxc auth identity list
```

The authentication method column displays the method by which the client authenticates with LXD.

The type column displays the type of identity. Identity types are a superset of TLS certificate types and additionally include OIDC clients.

The name column displays the name of the identity. For TLS clients, this will be the name of the certificate. For OIDC clients this will be the name of the client as given by the IdP (identity provider) (requested via the profile scope[205]).

The identifier column displays a unique identifier for the identity within that authentication method. For TLS clients, this will be the certificate fingerprint. For OIDC clients, this will be the email address of the client.

The groups column displays any groups that are currently assigned to the identity. Groups cannot currently be assigned to TLS clients.

Note

OIDC clients will only be displayed in the list of identities once they have authenticated with LXD.

Manage permissions

In LXD, identities cannot be granted permissions directly. Instead, identities are added to groups, and groups are granted permissions. To create a group, run:

```
lxc auth group create <group_name>
```

To add an identity to a group, run:

```
lxc auth identity group add <authentication_method>/<identifier> <group_name>
```

For example, for OIDC clients:

```
lxc auth identity group add oidc/<email_address> <group_name>
```

The identity is now a member of the group. To add permissions to the group, run:

```
lxc auth group permission add <group_name> <entity_type> [<entity_name>]
<entitlement> [<key>=<value>...]
```

Here are some examples:

- `lxc auth group permission add administrator server admin` grants members of `administrator` the `admin` entitlement on `server`.
- `lxc auth group permission add junior-dev project sandbox operator` grants members of `junior-dev` the `operator` entitlement on project `sandbox`.
- `lxc auth group permission add my-group instance c1 user project=default` grants members of `my-group` the `user` entitlement on instance `c1` in project `default`.

[205] https://openid.net/specs/openid-connect-basic-1_0.html#Scopes

Some entity types require more than one supplementary argument to uniquely specify the entity. For example, entities of type `storage_volume` and `storage_bucket` require an additional `pool=<storage_pool_name>` argument.

Use groups defined by the identity provider

It is common practice to manage users, roles, and groups centrally via an identity provider (IdP). In LXD, identity provider groups allow groups that are defined by the IdP to be mapped to LXD groups. When an OIDC client makes a request to LXD, any groups that can be extracted from the client's identity token are mapped to LXD groups, giving the client the same effective permissions.

To configure IdP group mappings in LXD, first configure your IdP to add groups to identity and access tokens as a custom claim. This configuration depends on your IdP. In [206], for example, you can add the "roles" that a user has as a custom claim via an action[207]. Alternatively, if RBAC (role-based access control) is enabled for the audience, a "permissions" claim can be added automatically. In Keycloak, you can define a mapper[208] to set Keycloak groups in the token.

Then configure LXD to extract this claim. To do so, set the value of the *`oidc.groups.claim`* (page 406) configuration key to the value of the field name of the custom claim:

```
lxc config set oidc.groups.claim=<claim_name>
```

LXD will then expect the identity and access tokens to contain a claim with this name. The value of the claim must be a JSON array containing a string value for each IdP group name. If the group names are extracted successfully, LXD will be aware of the IdP groups for the duration of the request.

Next, configure a mapping between an IdP group and a LXD group as follows:

```
lxc auth identity-provider-group create <idp_group_name>
lxc auth identity-provider-group group add <idp_group_name> <lxd_group_name>
```

IdP groups can be mapped to multiple LXD groups, and multiple IdP groups can be mapped to the same LXD group.

Important

LXD does not store the identity provider groups that are extracted from identity or access tokens. This can obfuscate the true permissions of an identity. For example, if an identity belongs to LXD group "foo", an administrator can view the permissions of group "foo" to determine the level of access of the identity. However, if identity provider group mappings are configured, direct group membership alone does not determine their level of access. The command `lxc auth identity info` can be run by any identity to view a full list of their own effective groups and permissions as granted directly or indirectly via IdP groups.

[206] https://auth0.com/
[207] https://community.auth0.com/t/how-to-add-roles-and-permissions-to-the-id-token-using-actions/84506
[208] https://keycloak.discourse.group/t/anyway-to-include-user-groups-into-my-jwt-token/8715

3.3.3. Instances grouping with projects

You can use projects to keep your LXD server clean by grouping related instances together. In addition to isolated instances, each project can also have specific images, profiles, networks, and storage.

For example, projects can be useful in the following scenarios:

- You run a huge number of instances for different purposes, for example, for different customer projects. You want to keep these instances separate to make it easier to locate and maintain them, and you might want to reuse the same instance names in each customer project for consistency reasons. Each instance in a customer project should use the same base configuration (for example, networks and storage), but the configuration might differ between customer projects.

  In this case, you can create a LXD project for each customer project (thus each group of instances) and use different profiles, networks, and storage for each LXD project.

- Your LXD server is shared between multiple users. Each user runs their own instances, and might want to configure their own profiles. You want to keep the user instances confined, so that each user can interact only with their own instances and cannot see the instances created by other users. In addition, you want to be able to limit resources for each user and make sure that the instances of different users cannot interfere with one another.

  In this case, you can set up a multi-user environment with confined projects.

LXD comes with a `default` project. See *How to create and configure projects* (page 161) for instructions on how to add projects.

Isolation of projects

Projects always encapsulate the instances they contain, which means that instances cannot be shared between projects and instance names can be duplicated in several projects. When you are in a specific project, you can see only the instances that belong to this project.

Other entities (images, profiles, networks, and storage) can be either isolated in the project or inherited from the `default` project. To configure which entities are isolated, you enable or disable the respective *feature* in the project. If a feature is enabled, the corresponding entity is isolated in the project; if the feature is disabled, it is inherited from the `default` project.

For example, if you enable *`features.networks`* (page 510) for a project, the project uses a separate set of networks and not the networks defined in the `default` project. If you disable *`features.images`* (page 510), the project has access to the images defined in the `default` project, and any images you add while you're using the project are also added to the `default` project.

See the list of available *Project features* (page 509) for information about which features are enabled or disabled when you create a project.

> **Note**
>
> You must select the features that you want to enable before starting to use a new project. When a project contains instances, the features are locked. To edit them, you must remove all instances first.

New features that are added in an upgrade are disabled for existing projects.

Important

In a multi-tenant environment, unless using *Fine-grained authorization* (page 364), all projects should have all features enabled. Otherwise, clients with *Restricted TLS certificates* (page 364) are able to create, edit, and delete resources in the default project. This might affect other tenants.

For example, if project "foo" is created and `features.networks` is not set to true, then a restricted client certificate with access to "foo" can view, edit, and delete networks in the default project.

Conversely, if a client's permissions are managed via *Fine-grained authorization* (page 364), resources may be inherited from the default project but access to those resources is not automatically granted.

Confined projects in a multi-user environment

If your LXD server is used by multiple users (for example, in a lab environment), you can use projects to confine the activities of each user. This method isolates the instances and other entities (depending on the feature configuration), as described in *Isolation of projects* (page 368). It also confines users to their own user space and prevents them from gaining access to other users' instances or data. Any changes that affect the LXD server and its configuration, for example, adding or removing storage, are not permitted.

In addition, this method allows users to work with LXD without being a member of the `lxd` group (see *Access to the LXD daemon* (page 377)). Members of the `lxd` group have full access to LXD, including permission to attach file system paths and tweak the security features of an instance, which makes it possible to gain root access to the host system. Using confined projects limits what users can do in LXD, but it also prevents users from gaining root access.

When LXD is accessible over the HTTPS API, both *TLS client certificates* (page 358) and *OIDC clients* (page 361) can be restricted to allow access to specific projects only. This is managed via *Fine-grained authorization* (page 364). See *Confine users to specific projects on the HTTPS API* (page 169) for instructions.

Multi-user LXD daemon

The LXD snap contains a multi-user LXD daemon that allows dynamic project creation on a per-user basis. You can configure a specific user group other than the `lxd` group to give restricted LXD access to every user in the group.

When a user that is a member of this group starts using LXD, the multi-user daemon automatically creates a confined project for this user.

If you're not using the snap, you can still use this feature if your distribution supports it.

See *Confine users to specific LXD projects via Unix socket* (page 175) for instructions on configuring the multi-user daemon.

Related topics

How-to guides:

- *Projects* (page 161)

Reference:

- *Project configuration* (page 509)

3.4. Production setup

When you're ready to move your LXD setup to production, you should read up on the concepts that are important for providing a scalable, reliable, and secure environment.

3.4.1. Clusters

To spread the total workload over several servers, LXD can be run in clustering mode. In this scenario, any number of LXD servers share the same distributed database that holds the configuration for the cluster members and their instances. The LXD cluster can be managed uniformly using the `lxc` (page 690) client or the REST API.

This feature was introduced as part of the `clustering` (page 633) API extension and is available since LXD 3.0.

> **Tip**
>
> If you want to quickly set up a basic LXD cluster, check out MicroCloud[209].

Cluster members

A LXD cluster consists of one bootstrap server and at least two further cluster members. It stores its state in a *distributed database* (page 356), which is a Dqlite[210] database replicated using the Raft algorithm.

While you could create a cluster with only two members, it is strongly recommended that the number of cluster members be at least three. With this setup, the cluster can survive the loss of at least one member and still be able to establish quorum for its distributed state.

When you create the cluster, the Dqlite database runs on only the bootstrap server until a third member joins the cluster. Then both the second and the third server receive a replica of the database.

See *How to form a cluster* (page 280) for more information.

Member roles

In a cluster with three members, all members replicate the distributed database that stores the state of the cluster. If the cluster has more members, only some of them replicate the database. The remaining members have access to the database, but don't replicate it.

At each time, there is an elected cluster leader that monitors the health of the other members.

Each member that replicates the database has either the role of a *voter* or of a *stand-by*. If the cluster leader goes offline, one of the voters is elected as the new leader. If a voter

[209] https://canonical.com/microcloud
[210] https://dqlite.io/

member goes offline, a stand-by member is automatically promoted to voter. The database (and hence the cluster) remains available as long as a majority of voters is online.

The following roles can be assigned to LXD cluster members. Automatic roles are assigned by LXD itself and cannot be modified by the user.

Role	Automatic	Description
`database`	yes	Voting member of the distributed database
`database-leader`	yes	Current leader of the distributed database
`database-standb`	yes	Stand-by (non-voting) member of the distributed database
`event-hub`	no	Exchange point (hub) for the internal LXD events (requires at least two)
`ovn-chassis`	no	Uplink gateway candidate for OVN networks

The default number of voter members (*cluster.max_voters* (page 408)) is three. The default number of stand-by members (*cluster.max_standby* (page 408)) is two. With this configuration, your cluster will remain operational as long as you switch off at most one voting member at a time.

See *How to manage a cluster* (page 284) for more information.

Offline members and fault tolerance

If a cluster member is down for more than the configured offline threshold, its status is marked as offline. In this case, no operations are possible on this member, and neither are operations that require a state change across all members.

As soon as the offline member comes back online, operations are available again.

If the member that goes offline is the leader itself, the other members will elect a new leader.

If you can't or don't want to bring the server back online, you can *delete it from the cluster* (page 287).

You can tweak the amount of seconds after which a non-responding member is considered offline by setting the *cluster.offline_threshold* (page 408) configuration. The default value is 20 seconds. The minimum value is 10 seconds.

To automatically *evacuate* (page 286) instances from an offline member, set the *cluster.healing_threshold* (page 407) configuration to a non-zero value.

See *How to recover a cluster* (page 293) for more information.

Failure domains

You can use failure domains to indicate which cluster members should be given preference when assigning roles to a cluster member that has gone offline. For example, if a cluster member that currently has the database role gets shut down, LXD tries to assign its database role to another cluster member in the same failure domain, if one is available.

To update the failure domain of a cluster member, use the *lxc cluster edit <member>* (page 717) command and change the `failure_domain` property from `default` to another string.

Member configuration

LXD cluster members are generally assumed to be identical systems. This means that all LXD servers joining a cluster must have an identical configuration to the bootstrap server, in terms of storage pools and networks.

To accommodate things like slightly different disk ordering or network interface naming, there is an exception for some configuration options related to storage and networks, which are member-specific.

When such settings are present in a cluster, any server that is being added must provide a value for them. Most often, this is done through the interactive `lxd init` command, which asks the user for the value for a number of configuration keys related to storage or networks.

Those settings typically include:

- The source device and size (quota) for a storage pool
- The name for a ZFS zpool, LVM thin pool or LVM volume group
- External interfaces and BGP next-hop for a bridged network
- The name of the parent network device for managed `physical` or `macvlan` networks

See *How to configure storage for a cluster* (page 290) and *How to configure networks for a cluster* (page 288) for more information.

If you want to look up the questions ahead of time (which can be useful for scripting), query the `/1.0/cluster` API endpoint. This can be done through `lxc query /1.0/cluster` or through other API clients.

Images

By default, LXD replicates images on as many cluster members as there are database members. This typically means up to three copies within the cluster.

You can increase that number to improve fault tolerance and the likelihood of the image being locally available. To do so, set the *`cluster.images_minimal_replica`* (page 407) configuration. The special value of `-1` can be used to have the image copied to all cluster members.

Cluster groups

In a LXD cluster, you can add members to cluster groups. You can use these cluster groups to launch instances on a cluster member that belongs to a subset of all available members. For example, you could create a cluster group for all members that have a GPU and then launch all instances that require a GPU on this cluster group.

By default, all cluster members belong to the `default` group.

See *How to set up cluster groups* (page 292) and *Launch an instance on a specific cluster member* (page 291) for more information.

Automatic placement of instances

In a cluster setup, each instance lives on one of the cluster members. When you launch an instance, you can target it to a specific cluster member, to a cluster group or have LXD automatically assign it to a cluster member.

By default, the automatic assignment picks the cluster member that has the lowest number of instances. If several members have the same amount of instances, one of the members is chosen at random.

However, you can control this behavior with the *scheduler.instance* (page 602) configuration option:

- If `scheduler.instance` is set to `all` for a cluster member, this cluster member is selected for an instance if:
  - The instance is created without `--target` and the cluster member has the lowest number of instances.
  - The instance is targeted to live on this cluster member.
  - The instance is targeted to live on a member of a cluster group that the cluster member is a part of, and the cluster member has the lowest number of instances compared to the other members of the cluster group.
- If `scheduler.instance` is set to `manual` for a cluster member, this cluster member is selected for an instance if:
  - The instance is targeted to live on this cluster member.
- If `scheduler.instance` is set to `group` for a cluster member, this cluster member is selected for an instance if:
  - The instance is targeted to live on this cluster member.
  - The instance is targeted to live on a member of a cluster group that the cluster member is a part of, and the cluster member has the lowest number of instances compared to the other members of the cluster group.

Instance placement scriptlet

LXD supports using custom logic to control automatic instance placement by using an embedded script (scriptlet). This method provides more flexibility than the built-in instance placement functionality.

The instance placement scriptlet must be written in the Starlark language[211] (which is a subset of Python). The scriptlet is invoked each time LXD needs to know where to place an instance. The scriptlet receives information about the instance that is being placed and the candidate cluster members that could host the instance. It is also possible for the scriptlet to request information about each candidate cluster member's state and the hardware resources available.

An instance placement scriptlet must implement the `instance_placement` function with the following signature:

```
instance_placement(request, candidate_members):
```

- `request` is an object that contains an expanded representation of `scriptlet.InstancePlacement`[212]. This request includes `project` and `reason` fields. The `reason` can be `new`, `evacuation` or `relocation`.

[211] https://github.com/bazelbuild/starlark
[212] https://pkg.go.dev/github.com/canonical/lxd/shared/api/scriptlet/#InstancePlacement

- candidate_members is a list of cluster member objects representing api.ClusterMember[213] entries.

For example:

```
def instance_placement(request, candidate_members):
    # Example of logging info, this will appear in LXD's log.
    log_info("instance placement started: ", request)

    # Example of applying logic based on the instance request.
    if request.name == "foo":
        # Example of logging an error, this will appear in LXD's log.
        log_error("Invalid name supplied: ", request.name)

        fail("Invalid name") # Exit with an error to reject instance placement.

    # Place the instance on the first candidate server provided.
    set_target(candidate_members[0].server_name)

    return # Return empty to allow instance placement to proceed.
```

The scriptlet must be applied to LXD by storing it in the *instances.placement.scriptlet* (page 412) global configuration setting.

For example, if the scriptlet is saved inside a file called `instance_placement.star`, then it can be applied to LXD with the following command:

```
cat instance_placement.star | lxc config set instances.placement.scriptlet=-
```

To see the current scriptlet applied to LXD, use the `lxc config get instances.placement.scriptlet` command.

The following functions are available to the scriptlet (in addition to those provided by Starlark):

- `log_info(*messages)`: Add a log entry to LXD's log at `info` level. `messages` is one or more message arguments.
- `log_warn(*messages)`: Add a log entry to LXD's log at `warn` level. `messages` is one or more message arguments.
- `log_error(*messages)`: Add a log entry to LXD's log at `error` level. `messages` is one or more message arguments.
- `set_cluster_member_target(member_name)`: Set the cluster member where the instance should be created. `member_name` is the name of the cluster member the instance should be created on. If this function is not called, then LXD will use its built-in instance placement logic.
- `get_cluster_member_state(member_name)`: Get the cluster member's state. Returns an object with the cluster member's state in the form of `api.ClusterMemberState`[214]. `member_name` is the name of the cluster member to get the state for.

[213] https://pkg.go.dev/github.com/canonical/lxd/shared/api#ClusterMember
[214] https://pkg.go.dev/github.com/canonical/lxd/shared/api#ClusterMemberState

- `get_cluster_member_resources(member_name)`: Get information about resources on the cluster member. Returns an object with the resource information in the form of `api.Resources`[215]. `member_name` is the name of the cluster member to get the resource information for.
- `get_instance_resources()`: Get information about the resources the instance will require. Returns an object with the resource information in the form of `scriptlet.InstanceResources`[216].

Note

Field names in the object types are equivalent to the JSON field names in the associated Go types.

Related topics

How-to guides:

- *Clustering* (page 280)

Reference:

- *Cluster member configuration* (page 602)

3.4.2. Performance tuning

When you are ready to move your LXD setup to production, you should take some time to optimize the performance of your system. There are different aspects that impact performance. The following steps help you to determine the choices and settings that you should tune to improve your LXD setup.

Run benchmarks

LXD provides a benchmarking tool to evaluate the performance of your system. You can use the tool to initialize or launch a number of containers and measure the time it takes for the system to create the containers. By running the tool repeatedly with different LXD configurations, system settings or even hardware setups, you can compare the performance and evaluate which is the ideal configuration.

See *How to benchmark performance* (page 297) for instructions on running the tool.

Monitor instance metrics

LXD collects metrics for all running instances as well as some internal metrics. These metrics cover the CPU, memory, network, disk and process usage. They are meant to be consumed by Prometheus, and you can use Grafana to display the metrics as graphs. See *Provided metrics* (page 606) for lists of available metrics and *Set up a Grafana dashboard* (page 309) for instructions on how to display the metrics in Grafana.

You should regularly monitor the metrics to evaluate the resources that your instances use. The numbers help you to determine if there are any spikes or bottlenecks, or if usage patterns change and require updates to your configuration.

See *How to monitor metrics* (page 301) for more information about metrics collection.

[215] https://pkg.go.dev/github.com/canonical/lxd/shared/api#Resources
[216] https://pkg.go.dev/github.com/canonical/lxd/shared/api/scriptlet/#InstanceResources

Tune server settings

The default kernel settings for most Linux distributions are not optimized for running a large number of containers or virtual machines. Therefore, you should check and modify the relevant server settings to avoid hitting limits caused by the default settings.

Typical errors that you might see when you encounter those limits are:

- `Failed to allocate directory watch: Too many open files`
- `<Error> <Error>: Too many open files`
- `failed to open stream: Too many open files in...`
- `neighbour: ndisc_cache: neighbor table overflow!`

See *Server settings for a LXD production setup* (page 602) for a list of relevant server settings and suggested values.

Tune the network bandwidth

If you have a lot of local activity between instances or between the LXD host and the instances, or if you have a fast internet connection, you should consider increasing the network bandwidth of your LXD setup. You can do this by increasing the transmit and receive queue lengths.

See *How to increase the network bandwidth* (page 299) for instructions.

Related topics

How-to guides:

- *How to benchmark performance* (page 297)
- *How to increase the network bandwidth* (page 299)
- *How to monitor metrics* (page 301)

Reference:

- *Provided metrics* (page 606)
- *Server settings for a LXD production setup* (page 602)

3.4.3. Security

Consider the following aspects to ensure that your LXD installation is secure:

- Keep your operating system up-to-date and install all available security patches.
- Use only supported LXD versions (LTS releases or the latest feature release).
- Restrict access to the LXD daemon and the remote API.
- Configure your network interfaces to be secure.
- Do not use privileged containers unless required. If you use privileged containers, put appropriate security measures in place.

See the following sections for detailed information.

If you discover a security issue, see the LXD security policy[217] for information on how to report the issue.

[217] https://github.com/canonical/lxd/blob/main/SECURITY.md

Supported versions

Never use unsupported LXD versions in a production environment.

LXD has two types of releases:

- Feature releases
- LTS releases

For feature releases, only the latest one is supported, and we usually don't do point releases. Instead, users are expected to wait until the next feature release.

For LTS releases, we do periodic bugfix releases that include an accumulation of bugfixes from the feature releases. Such bugfix releases do not include new features.

Access to the LXD daemon

LXD is a daemon that can be accessed locally over a Unix socket or, if configured, remotely over a TLS socket. Anyone with access to the socket can fully control LXD, which includes the ability to attach host devices and file systems or to tweak the security features for all instances.

Therefore, make sure to restrict the access to the daemon to trusted users.

Local access to the LXD daemon

The LXD daemon runs as root and provides a Unix socket for local communication. Access control for LXD is based on group membership. The root user and all members of the `lxd` group can interact with the local daemon.

> **Important**
>
> Local access to LXD through the Unix socket always grants full access to LXD. This includes the ability to attach file system paths or devices to any instance as well as tweak the security features on any instance.
>
> Therefore, you should only give such access to users who you'd trust with root access to your system.

Access to the remote API

By default, access to the daemon is only possible locally. By setting the *`core.https_address`* (page 402) configuration option, you can expose the same API over the network on a TLS socket. See *How to expose LXD to the network* (page 44) for instructions. Remote clients can then connect to LXD and access any image that is marked for public use.

There are several ways to authenticate remote clients as trusted clients to allow them to access the API. See *Remote API authentication* (page 358) for details.

In a production setup, you should set *`core.https_address`* (page 402) to the single address where the server should be available (rather than any address on the host). In addition, you should set firewall rules to allow access to the LXD port only from authorized hosts/subnets.

Container security

LXD containers can use a wide range of features for security.

Also see the LXC security page[218] on `linuxcontainers.org` for details on LXC container security and the applied kernel features.

Unprivileged containers

By default, containers are *unprivileged*, meaning that they operate inside a user namespace, restricting the abilities of users in the container to that of regular users on the host with limited privileges on the devices that the container owns.

Unprivileged containers are safe by design: The container UID 0 is mapped to an unprivileged user outside of the container. It has extra rights only on resources that it owns itself.

This mechanism ensures that most security issues (for example, container escape or resource abuse) that might occur in a container apply just as well to a random unprivileged user, which means they are a generic kernel security bug rather than a LXD issue.

> **Tip**
>
> If data sharing between containers isn't needed, you can enable *`security.idmap.isolated`* (page 436), which will use non-overlapping UID/GID maps for each container, preventing potential DoS (Denial of Service) attacks on other containers.

Privileged containers

LXD can also run *privileged* containers. In privileged containers, the container UID 0 is mapped to the host's UID 0.

Such privileged containers are not root-safe, and a user with root access in such a container will be able to DoS the host as well as find ways to escape confinement.

LXC applies some protection measures to privileged containers to prevent accidental damage of the host (where damage is defined as things like reconfiguring host hardware, reconfiguring the host kernel, or accessing the host file system). This protection of the host and prevention of escape is achieved through mandatory access control (`apparmor`, `selinux`), Seccomp filters, dropping of capabilities, and namespaces. These measures are valuable when running trusted workloads, but they do not make privileged containers root-safe.

Therefore, you should not use privileged containers unless required. If you use them, make sure to put appropriate security measures in place.

Container name leakage

The default server configuration makes it easy to list all cgroups on a system and, by extension, all running containers.

You can prevent this name leakage by blocking access to `/sys/kernel/slab` and `/proc/sched_debug` before you start any containers. To do so, run the following commands:

[218] https://linuxcontainers.org/lxc/security/

```
chmod 400 /proc/sched_debug
chmod 700 /sys/kernel/slab/
```

Network security

Make sure to configure your network interfaces to be secure. Which aspects you should consider depends on the networking mode you decide to use.

Bridged NIC security

The default networking mode in LXD is to provide a "managed" private network bridge that each instance connects to. In this mode, there is an interface on the host called `lxdbr0` that acts as the bridge for the instances.

The host runs an instance of `dnsmasq` for each managed bridge, which is responsible for allocating IP addresses and providing both authoritative and recursive DNS services.

Instances using DHCPv4 will be allocated an IPv4 address, and a DNS record will be created for their instance name. This prevents instances from being able to spoof DNS records by providing false host name information in the DHCP request.

The `dnsmasq` service also provides IPv6 router advertisement capabilities. This means that instances will auto-configure their own IPv6 address using SLAAC, so no allocation is made by `dnsmasq`. However, instances that are also using DHCPv4 will also get an AAAA DNS record created for the equivalent SLAAC IPv6 address. This assumes that the instances are not using any IPv6 privacy extensions when generating IPv6 addresses.

In this default configuration, whilst DNS names cannot not be spoofed, the instance is connected to an Ethernet bridge and can transmit any layer 2 traffic that it wishes, which means an instance that is not trusted can effectively do MAC or IP spoofing on the bridge.

In the default configuration, it is also possible for instances connected to the bridge to modify the LXD host's IPv6 routing table by sending (potentially malicious) IPv6 router advertisements to the bridge. This is because the `lxdbr0` interface is created with `/proc/sys/net/ipv6/conf/lxdbr0/accept_ra` set to `2`, meaning that the LXD host will accept router advertisements even though `forwarding` is enabled (see `/proc/sys/net/ipv4/*` Variables[219] for more information).

However, LXD offers several bridged NIC security features that can be used to control the type of traffic that an instance is allowed to send onto the network. These NIC settings should be added to the profile that the instance is using, or they can be added to individual instances, as shown below.

The following security features are available for bridged NICs:

[219] https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt

Key	Type	Default	Required	Description
`security.mac_filtering`	bool	`false`	no	Prevent the instance from spoofing another instance's MAC address
`security.ipv4_filtering`	bool	`false`	no	Prevent the instance from spoofing another instance's IPv4 address (enables `mac_filtering`)
`security.ipv6_filtering`	bool	`false`	no	Prevent the instance from spoofing another instance's IPv6 address (enables `mac_filtering`)

One can override the default bridged NIC settings from the profile on a per-instance basis using:

```
lxc config device override <instance> <NIC> security.mac_filtering=true
```

Used together, these features can prevent an instance connected to a bridge from spoofing MAC and IP addresses. These options are implemented using either `xtables` (`iptables`, `ip6tables` and `ebtables`) or `nftables`, depending on what is available on the host.

It's worth noting that those options effectively prevent nested containers from using the parent network with a different MAC address (i.e using bridged or `macvlan` NICs).

The IP filtering features block ARP and NDP advertisements that contain a spoofed IP, as well as blocking any packets that contain a spoofed source address.

If `security.ipv4_filtering` or `security.ipv6_filtering` is enabled and the instance cannot be allocated an IP address (because `ipvX.address=none` or there is no DHCP service enabled on the bridge), then all IP traffic for that protocol is blocked from the instance.

When `security.ipv6_filtering` is enabled, IPv6 router advertisements are blocked from the instance.

When `security.ipv4_filtering` or `security.ipv6_filtering` is enabled, any Ethernet frames that are not ARP, IPv4 or IPv6 are dropped. This prevents stacked VLAN Q-in-Q (802.1ad) frames from bypassing the IP filtering.

Routed NIC security

An alternative networking mode is available called "routed". It provides a virtual Ethernet device pair between container and host. In this networking mode, the LXD host functions as a router, and static routes are added to the host directing traffic for the container's IPs towards the container's `veth` interface.

By default, the `veth` interface created on the host has its `accept_ra` setting disabled to prevent router advertisements from the container modifying the IPv6 routing table on the LXD host. In addition to that, the `rp_filter` on the host is set to `1` to prevent source address spoofing for IPs that the host does not know the container has.

Related topics

How-to guides:

- *How to expose LXD to the network* (page 44)

Explanation:

- *Remote API authentication* (page 358)

3.4.4. Privilege delegation using BPF Token

Overview

The *security.delegate_bpf* (page 434) option enables the BPF (Berkeley Packet Filter) functionality delegation mechanism, using a BPF Token[220]. When enabled, LXD mounts a BPF File System (BPFFS) inside a container instance. This file system is configured with the `security.delegate_bpf.*` settings. For example:

```
none on /sys/fs/bpf type bpf (rw,relatime,uid=1000000,gid=1000000,
                              delegate_cmds=map_create:prog_load,
                              delegate_maps=ringbuf,
                              delegate_progs=socket_filter,
                              delegate_attachs=cgroup_inet_ingress)
```

Then, applications inside the container can create a BPF Token file descriptor using that BPFFS mount and the `bpf(BPF_TOKEN_CREATE)` syscall. Later, this File Descriptor can be passed to `bpf(BPF_PROG_LOAD)`, `bpf(BPF_MAP_CREATE)`, or another `bpf()`-command syscall, and the kernel will perform a permission check against the token instead of the current user credentials. To be more precise, current user caps are also checked for `CAP_BPF` but in a current user namespace when `bpf(BPF_TOKEN_CREATE)` is called.

It follows that user space applications inside the container must be aware of the BPF Token kernel feature (which appeared in Linux kernel v6.9) and make use of it. In contrast to `security.syscalls.intercept.*` features, this one is not fully transparent and might require updates or modifications to the software inside the container. Fortunately, the libbpf library[221] supports BPF tokens. Thus if an application uses libbpf, then to make use of this feature, you might only need to update libbpf.

> **Note**
>
> Configure the following instance options for the container, depending on its BPF workload:
>
> - *security.delegate_bpf.cmd_types* (page 434)
> - *security.delegate_bpf.map_types* (page 434)
> - *security.delegate_bpf.prog_types* (page 435)
> - *security.delegate_bpf.attach_types* (page 434)

See the BPF Token documentation page[222] on `docs.ebpf.io` for details.

Example (socket filter)

Let's consider an example with a socket filter program from libbpf-bootstrap[223].

The following creates an unprivileged container instance and sets all the necessary configuration options to enable BPF delegation:

220 https://docs.ebpf.io/linux/concepts/token
221 https://docs.kernel.org/bpf/libbpf/libbpf_overview.html
222 https://docs.ebpf.io/linux/concepts/token/
223 https://github.com/libbpf/libbpf-bootstrap

```
lxc launch ubuntu:noble bpf-experimentslxc config set bpf-experiments
limits.kernel.memlock=unlimitedlxc config set bpf-experiments
security.delegate_bpf=truelxc config set bpf-experiments
security.delegate_bpf.prog_types=socket_filterlxc config set bpf-experiments
security.delegate_bpf.attach_types=cgroup_inet_ingresslxc config set
bpf-experiments security.delegate_bpf.cmd_types=prog_load:map_createlxc
config set bpf-experiments security.delegate_bpf.map_types=ringbuf
```

The following set of commands clones and builds the libbpf-bootstrap.git repository within the example `bpf-experiments` container:

```
lxc shell bpf-experimentsapt install clang build-essentialgit clone
https://github.com/libbpf/libbpf-bootstrap.gitgit submodule update --init
--recursivecd libbpf-bootstrap/examples/cmake
```

This experiment completes by running commands from two different shells into the `bpf-experiments` container.

From one terminal:

```
lxc shell bpf-experiments./sockfilter
```

From another terminal:

```
lxc shell bpf-experimentsping -c 4 localhost
```

Sample output:

```
ibbpf: loading object 'sockfilter_bpf' from buffer
libbpf: elf: section(2) .symtab, size 192, link 1, flags 0, type=2
libbpf: elf: section(3) socket, size 576, link 0, flags 6, type=1
libbpf: sec 'socket': found program 'socket_handler' at insn offset 0 (0 bytes),
code size 72 insns (576 bytes)
...
libbpf: Kernel doesn't support BTF, skipping uploading it.
libbpf: map 'rb': created successfully, fd=3
interface: lo        protocol: ICMP        127.0.0.1:2048(src) -> 127.0.0.
1:32429(dst)
interface: lo        protocol: ICMP        127.0.0.1:0(src) -> 127.0.0.
1:34477(dst)
interface: lo        protocol: ICMP        127.0.0.1:2048(src) -> 127.0.0.
1:46163(dst)
interface: lo        protocol: ICMP        127.0.0.1:0(src) -> 127.0.0.
1:48211(dst)
```

We can see from this sample output that the ICMP packets were captured by the eBPF (extended Berkeley Capture Filter) program and logged.

Finding the right configuration

To figure out the right values for the `security.delegate_bpf.cmd_types`, `security.delegate_bpf.map_types`, `security.delegate_bpf.prog_types`, `security.delegate_bpf.attach_types` options, you must know how your application inside the container uses eBPF, such as its program types and map types. You can consult the application's source code, or use the `strace`[224] tool to trace `bpf` syscall and see how it is being used.

Example using `strace`:

```
strace -e bpf ./sockfilter
```

Sample output:

```
bpf(0x24 /* BPF_??? */, 0x7fffafdf5a40, 8) = 5
bpf(BPF_PROG_LOAD, {prog_type=BPF_PROG_TYPE_SOCKET_FILTER, insn_cnt=2,
insns=0x7fffafdf59e0, license="GPL", log_level=0, log_size=0, log_buf=NULL, kern_
version=KERNEL_VERSION(0, 0, 0), prog_flags=0, prog_name="", prog_ifindex=0,
expected_attach_type=BPF_CGROUP_INET_INGRESS, prog_btf_fd=0, func_info_rec_size=0,
func_info=NULL, func_info_cnt=0, line_info_rec_size=0, line_info=NULL, line_info_
cnt=0, attach_btf_id=0, attach_prog_fd=0, fd_array=NULL}, 148) = -1 EPERM
(Operation not permitted)
bpf(BPF_PROG_LOAD, {prog_type=BPF_PROG_TYPE_SOCKET_FILTER, insn_cnt=2,
insns=0x7fffafdf5c10, license="GPL", log_level=0, log_size=0, log_buf=NULL, kern_
version=KERNEL_VERSION(0, 0, 0), prog_flags=0x10000 /* BPF_F_??? */, prog_name="",
prog_ifindex=0, expected_attach_type=BPF_CGROUP_INET_INGRESS, prog_btf_fd=0, func_
info_rec_size=0, func_info=NULL, func_info_cnt=0, line_info_rec_size=0, line_
info=NULL, line_info_cnt=0, attach_btf_id=0, attach_prog_fd=0, fd_array=NULL, ...}
, 152) = 4
bpf(BPF_BTF_LOAD, {btf="\237\353\1\0\30\0\0\0\0\0\0\0000\0\0\0000\0\0\0\t\0\0\0\1\
0\0\0\0\0\0\1"..., btf_log_buf=NULL, btf_size=81, btf_log_size=0, btf_log_level=0,
...}, 40) = -1 EPERM (Operation not permitted)
bpf(BPF_BTF_LOAD, {btf="\237\353\1\0\30\0\0\0\0\0\0\0000\0\0\0000\0\0\0\5\0\0\0\0\
0\0\0\0\0\0\1"..., btf_log_buf=NULL, btf_size=77, btf_log_size=0, btf_log_level=0,
...}, 40) = -1 EPERM (Operation not permitted)
bpf(BPF_BTF_LOAD, {btf="\237\353\1\0\30\0\0\0\0\0\0\0\20\0\0\0\20\0\0\0\5\0\0\0\1\
0\0\0\0\0\0\1"..., btf_log_buf=NULL, btf_size=45, btf_log_size=0, btf_log_level=0,
...}, 40) = -1 EPERM (Operation not permitted)
libbpf: Kernel doesn't support BTF, skipping uploading it.
bpf(BPF_PROG_LOAD, {prog_type=BPF_PROG_TYPE_SOCKET_FILTER, insn_cnt=2,
insns=0x7fffafdf59c0, license="GPL", log_level=0, log_size=0, log_buf=NULL, kern_
version=KERNEL_VERSION(0, 0, 0), prog_flags=0x10000 /* BPF_F_??? */, prog_name=
"libbpf_nametest", prog_ifindex=0, expected_attach_type=BPF_CGROUP_INET_INGRESS,
prog_btf_fd=0, func_info_rec_size=0, func_info=NULL, func_info_cnt=0, line_info_
rec_size=0, line_info=NULL, line_info_cnt=0, attach_btf_id=0, attach_prog_fd=0,
fd_array=NULL, ...}, 148) = 4
bpf(BPF_PROG_LOAD, {prog_type=BPF_PROG_TYPE_SOCKET_FILTER, insn_cnt=2,
insns=0x7fffafdf58e0, license="GPL", log_level=0, log_size=0, log_buf=NULL, kern_
version=KERNEL_VERSION(0, 0, 0), prog_flags=0, prog_name="libbpf_nametest", prog_
ifindex=0, expected_attach_type=BPF_CGROUP_INET_INGRESS, prog_btf_fd=0, func_info_
```

(continues on next page)

[224] https://github.com/strace/strace

(continued from previous page)

```
rec_size=0, func_info=NULL, func_info_cnt=0, line_info_rec_size=0, line_info=NULL,
line_info_cnt=0, attach_btf_id=0, attach_prog_fd=0, fd_array=NULL}, 148) = -1
EPERM (Operation not permitted)
bpf(BPF_MAP_CREATE, {map_type=BPF_MAP_TYPE_RINGBUF, key_size=0, value_size=0, max_
entries=262144, map_flags=0x10000 /* BPF_F_??? */, inner_map_fd=0, map_name="",
map_ifindex=0, btf_fd=0, btf_key_type_id=0, btf_value_type_id=0, btf_vmlinux_
value_type_id=0, map_extra=0, ...}, 80) = 4
libbpf: map 'rb': created successfully, fd=3
bpf(BPF_PROG_LOAD, {prog_type=BPF_PROG_TYPE_SOCKET_FILTER, insn_cnt=72,
insns=0x56198b83c180, license="Dual BSD/GPL", log_level=0, log_size=0, log_
buf=NULL, kern_version=KERNEL_VERSION(6, 12, 14), prog_flags=0x10000 /* BPF_F_???
*/, prog_name="", prog_ifindex=0, expected_attach_type=BPF_CGROUP_INET_INGRESS,
prog_btf_fd=0, func_info_rec_size=0, func_info=NULL, func_info_cnt=0, line_info_
rec_size=0, line_info=NULL, line_info_cnt=0, attach_btf_id=0, attach_prog_fd=0,
fd_array=NULL, ...}, 152) = 4
bpf(BPF_OBJ_GET_INFO_BY_FD, {info={bpf_fd=3, info_len=88, info=0x7fffafdf5de0}},
16) = 0
```

This log shows that `sockfilter` is using:

1. Program types: `BPF_PROG_TYPE_SOCKET_FILTER`
2. Map types: `BPF_MAP_TYPE_RINGBUF`
3. Attachment types: `BPF_CGROUP_INET_INGRESS`
4. BPF commands: `BPF_BTF_LOAD`, `BPF_PROG_LOAD`, `BPF_MAP_CREATE`

4. Reference

The reference material in this section provides technical descriptions of LXD.

4.1. General information

Before you start using LXD, you should check the system requirements. You should also be aware of the supported architectures, its release types and snap information, the available image servers, the format for images, and the environment used for containers.

4.1.1. Requirements

Go

LXD requires Go 1.24.4 or higher and is only tested with the Golang compiler.

We recommend having at least 2GiB of RAM to allow the build to complete.

Kernel requirements

The minimum supported kernel version is 5.15, but older kernels should also work to some degree.

LXD requires a kernel with support for:

- Namespaces (`pid`, `net`, `uts`, `ipc` and `mount`)
- Seccomp
- Native Linux AIO (`io_setup(2)`[225], etc.)

The following optional features also require extra kernel options or newer versions:

- Namespaces (`user` and `cgroup`)
- AppArmor (including Ubuntu patch for mount mediation)
- Control Groups (`blkio`, `cpuset`, `devices`, `memory`, `pids` and `net_prio`)
- CRIU (exact details to be found with CRIU upstream)
- SKBPRIO/QFQ qdiscs (for `limits.priority`, minimum kernel 5.17)

As well as any other kernel feature required by the LXC version in use.

LXC

LXD requires LXC 5.0.0 or higher with the following build options:

- `apparmor` (if using LXD's AppArmor support)
- `seccomp`

To run recent version of various distributions, including Ubuntu, LXCFS should also be installed.

[225] https://man7.org/linux/man-pages/man2/io_setup.2.html

QEMU

For virtual machines, QEMU 6.2 or higher and `virtiofsd` 1.10.0 or higher are required. Some features like Confidential Guest support require a more recent QEMU and kernel version.

Hardware-assisted virtualization (Intel VT-x, AMD-V, etc) is required for running virtual machines. Additional hardware support (Intel VT-d, AMD-Vi) may be required for device passthrough.

ZFS

For the ZFS storage driver, ZFS 2.1 or higher is required. Some features like `zfs_delegate` requires 2.2 or higher to be used.

Additional libraries (and development headers)

LXD uses `dqlite` for its database, to build and set it up, you can run `make deps`.

LXD itself also uses a number of (usually packaged) C libraries:

- `libacl1`
- `libcap2`
- `liblz4` (for `dqlite`)
- `libuv1` (for `dqlite`)
- `libsqlite3` >= 3.37.2 (for `dqlite`)

Make sure you have all these libraries themselves and their development headers (`-dev` packages) installed.

Related topics

Tutorials:

- *First steps with LXD* (page 4)
- *Getting started with the UI* (page 11)

How-to guides:

- *Getting started* (page 28)

4.1.2. Architectures

LXD can run on just about any architecture that is supported by the Linux kernel and by Go.

Some entities in LXD are tied to an architecture, for example, the instances, instance snapshots and images.

The following table lists all supported architectures including their unique identifier and the name used to refer to them. The architecture names are typically aligned with the Linux kernel architecture names.

ID	Kernel name	Description	Personalities
1	`i686`	32bit Intel x86	
2	`x86_64`	64bit Intel x86	`x86`
3	`armv7l`	32bit ARMv7 little-endian	
4	`aarch64`	64bit ARMv8 little-endian	`armv7l` (optional)
5	`ppc`	32bit PowerPC big-endian	
6	`ppc64`	64bit PowerPC big-endian	`powerpc`
7	`ppc64le`	64bit PowerPC little-endian	
8	`s390x`	64bit ESA/390 big-endian	
9	`mips`	32bit MIPS	
10	`mips64`	64bit MIPS	`mips`
11	`riscv32`	32bit RISC-V little-endian	
12	`riscv64`	64bit RISC-V little-endian	
13	`armv6l`	32bit ARMv6 little-endian	
14	`armv8l`	32bit ARMv8 little-endian	
15	`loongarch64`	64bit LoongArch	

Note

LXD cares only about the kernel architecture, not the particular userspace flavor as determined by the toolchain.

That means that LXD considers ARMv7 hard-float to be the same as ARMv7 soft-float and refers to both as `armv7l`. If useful to the user, the exact userspace ABI may be set as an image and container property, allowing easy query.

Virtual machine support

LXD only supports running virtual machines on the following host architectures:

- `x86_64`
- `aarch64`
- `ppc64le`
- `s390x`

The virtual machine guest architecture can usually be the 32bit personality of the host architecture, so long as the virtual machine firmware is capable of booting it.

4.1.3. Releases and snap

Releases

The LXD team maintains both Long Term Support (LTS) and feature releases in parallel. Release notes are published on Discourse[226].

LTS releases

LTS releases are **intended for production use**.

LXD follows the Ubuntu release cycle[227] cadence, meaning that an LTS release of LXD is created every two years. The release names follow the format *x.y.z*, always including the point number *z*. Updates are provided through point releases, incrementing *z*.

Support

LTS releases receive standard support for five years, meaning that it receives continuous updates according to the support levels described below. An Ubuntu Pro[228] subscription can provide additional support and extends the support duration by an additional five years.

Support levels

Standard support for an LTS release starts at full support for its first two years, then moves to maintenance support for the remaining three years. Once an LTS reaches End of Life (EOL), it no longer receives any updates.

- **Full support**: Some new features, frequent bugfixes, and security updates are provided every six months. This schedule is an estimate that can change based on priorities and discovered bugs.
- **Maintenance support**: High impact bugfixes and critical security updates are provided as needed.

Currently supported

The currently supported LTS releases are 5.21.*z* and 5.0.*z*.

- 5.21.*z* is supported until June 2029.
  - Currently in full support phase.
- 5.0.*z* is supported until June 2027.
  - Currently in maintenance support phase.

Feature releases

Feature releases are pushed out more often and contain the newest features and bugfixes. Since they are less tested than LTS releases, they are **not recommended for production use**.

These releases follow the format *x.y*, and they never include a point number *z*. Currently, feature releases for LXD are numbered 6.*y*, with *y* incrementing for each new release. Every two years, the latest feature release becomes an LTS release.

[226] https://discourse.ubuntu.com/tags/c/lxd/news/143/release
[227] https://ubuntu.com/about/release-cycle
[228] https://ubuntu.com/pro

Support

Feature releases receive continuous updates via each new release. The newest release at any given time is also eligible for additional support through an Ubuntu Pro[229] subscription.

The LXD snap

The recommended way to *install LXD* (page 28) is its snap package[230], if snaps are available for your system. A key benefit of snap packaging is that it includes all required dependencies. This allows LXD to run in a consistent environment on many different Linux distributions. Using the snap also streamlines updates through its channels[231].

Channels

Each installed LXD snap follows a channel[232]. Channels are composed of a *track* (page 389) and a *risk level* (page 390) (for example, the 6/stable channel). Each channel points to one release at a time, and when a new release is published to a channel, it replaces the previous one. *Updating the snap* (page 321) then updates to that release.

To view all available channels, run:

```
snap info lxd
```

Tracks

LXD releases are grouped under snap tracks[233], such as 6 or 5.21.

LTS tracks

LXD LTS tracks use the format *x[.y]*, corresponding to the major and minor numbers of *LTS releases* (page 388).

Tracks up to `5.21` include both *x* and *y*, but future LTS tracks will use only *x*.

Feature track

The LXD feature track uses the major number of the current *feature release* (page 388). The current feature track is 6.

Feature releases within the same major version are published to the same track, replacing the previous release. For example, the `6.4` release replaced `6.3` in the `6` track. This simplifies updates, as you don't need to switch channels to access new feature releases within the same major version.

Every two years, the current feature track becomes the next LTS, and a new feature track is then created by incrementing *x*. For example, after the `6` track becomes an LTS, the `7` track is created and becomes the next feature track.

[229] https://ubuntu.com/pro
[230] https://snapcraft.io/lxd
[231] https://snapcraft.io/docs/channels
[232] https://snapcraft.io/docs/channels
[233] https://snapcraft.io/docs/channels#heading--tracks

The default track

If you *install the LXD snap* (page 28) without specifying a track, the recommended default is used. The default track always points to the most recent LTS track, which is currently 5.21.

The `latest` track

In the list of channels shown by `snap info lxd`, you might see channels with a track named `latest`. This track typically points to the latest feature release.

Since `latest` is a continuously rolling release track, it might become incompatible with your host OS version over time. Due to this, this track is *not recommended for general use* and might be removed in the future. Instead, use a feature or LTS track.

Risk levels

For each LXD track, there are three risk levels[234]: `stable`, `candidate`, and `edge`.

We recommend that you use the `stable` risk level to install fully tested releases; this is the only risk level supported under Ubuntu Pro[235], as well as the default risk level if one is not specified at install. The `candidate` and `edge` levels offer newer but less-tested updates, posing higher risk.

Updates

By default, installed snaps update automatically when new releases are published to the channel they're tracking. For control over LXD updates, we recommend that you modify this auto-update behavior by either *holding* (page 322) or *scheduling updates* (page 322) as described in our *How to manage the LXD snap* (page 321) guide. You can then apply updates according to your needs.

Updates on clusters

New LXD releases are published progressively as snaps[236]. This means that updates might not be immediately available to all machines at the same time.

This can cause issues when updating the LXD snap for *clusters* (page 370), as cluster members must use the same version of the snap at all times. For a guide on how to avoid this issue using the `--cohort` flag, see *Synchronize updates for a LXD cluster cohort* (page 323).

Related topics

How-to guides:

- *How to get support* (page 334)
- *Install the LXD snap package* (page 28)
- *How to manage the LXD snap* (page 321)

[234] https://snapcraft.io/docs/channels#heading--risk-levels
[235] https://ubuntu.com/pro
[236] https://documentation.ubuntu.com/snapcraft/stable/how-to/publishing/manage-revisions-and-releases/#deliver-a-progressive-release

4.1.4. Remote image servers

The *lxc* (page 690) CLI command comes pre-configured with the following default remote image servers:

images:
: This server provides unofficial images for a variety of Linux distributions. The images are built to be compact and minimal, and therefore the default image variants do not include `cloud-init`. Where possible, `/cloud` variants that include `cloud-init` are provided. See *cloud-init support in images* (page 116).

  This server does not provide official Ubuntu images (for those, use the `ubuntu:` server). It does, however, provide desktop variants of current Ubuntu releases.

  See `images.lxd.canonical.com`[237] for an overview of available images.

ubuntu:
: This server provides official stable Ubuntu images. All images are cloud images, which means that they include both `cloud-init` and the `lxd-agent`.

  See `cloud-images.ubuntu.com/releases`[238] for an overview of available images.

ubuntu-daily:
: This server provides official daily Ubuntu images. All images are cloud images, which means that they include both `cloud-init` and the `lxd-agent`.

  See `cloud-images.ubuntu.com/daily`[239] for an overview of available images.

ubuntu-minimal:
: This server provides official Ubuntu Minimal images. All images are cloud images, which means that they include both `cloud-init` and the `lxd-agent`.

  See `cloud-images.ubuntu.com/minimal/releases`[240] for an overview of available images.

ubuntu-minimal-daily:
: This server provides official daily Ubuntu Minimal images. All images are cloud images, which means that they include both `cloud-init` and the `lxd-agent`.

  See `cloud-images.ubuntu.com/minimal/daily`[241] for an overview of available images.

Remote server types

LXD supports the following types of remote image servers:

Simple streams servers
: Pure image servers that use the simple streams format[242]. The default image servers are simple streams servers.

Public LXD servers
: LXD servers that are used solely to serve images and do not run instances themselves.

[237] https://images.lxd.canonical.com
[238] https://cloud-images.ubuntu.com/releases/
[239] https://cloud-images.ubuntu.com/daily/
[240] https://cloud-images.ubuntu.com/minimal/releases/
[241] https://cloud-images.ubuntu.com/minimal/daily/
[242] https://git.launchpad.net/simplestreams/tree/

To make a LXD server publicly available over the network on port 8443, set the *core.https_address* (page 402) configuration option to :8443 and do not configure any authentication methods (see *How to expose LXD to the network* (page 44) for more information). Then set the images that you want to share to `public`.

LXD servers

Regular LXD servers that you can manage over a network, and that can also be used as image servers.

For security reasons, you should restrict the access to the remote API and configure an authentication method to control access. See *How to expose LXD to the network* (page 44) and *Remote API authentication* (page 358) for more information.

Related topics

How-to guides:

- *Images* (page 148)

Explanation:

- *Local and remote images* (page 348)

4.1.5. Image format

Images contain a root file system and a metadata file that describes the image. They can also contain templates for creating files inside an instance that uses the image.

Images can be packaged as either a unified image (single file) or a split image (two files).

Content

Images for containers have the following directory structure:

```
metadata.yaml
rootfs/
templates/
```

Images for VMs have the following directory structure:

```
metadata.yaml
rootfs.img
templates/
```

For both instance types, the `templates/` directory is optional.

Metadata

The `metadata.yaml` file contains information that is relevant to running the image in LXD. It includes the following information:

```yaml
architecture: x86_64
creation_date: 1424284563
properties:
  description: Ubuntu 24.04 LTS Intel 64bit
  os: Ubuntu
```

(continues on next page)

(continued from previous page)

```
  release: noble 24.04
templates:
  ...
```

The architecture and creation_date fields are mandatory. The properties field contains a set of default properties for the image. The os, release, name and description fields are commonly used, but are not mandatory.

The templates field is optional. See *Templates (optional)* (page 393) for information on how to configure templates.

Root file system

For containers, the rootfs/ directory contains a full file system tree of the root directory (/) in the container.

Virtual machines use a rootfs.img qcow2 file instead of a rootfs/ directory. This file becomes the main disk device.

Templates (optional)

You can use templates to dynamically create files inside an instance. To do so, configure template rules in the metadata.yaml file and place the template files in a templates/ directory.

As a general rule, you should never template a file that is owned by a package or is otherwise expected to be overwritten by normal operation of an instance.

Template rules

For each file that should be generated, create a rule in the metadata.yaml file. For example:

```
templates:
  /etc/hosts:
    when:
      - create
      - rename
    template: hosts.tpl
    properties:
      foo: bar
  /etc/hostname:
    when:
      - start
    template: hostname.tpl
  /etc/network/interfaces:
    when:
      - create
    template: interfaces.tpl
    create_only: true
```

The when key can be one or more of:

- create - run at the time a new instance is created from the image

- `copy` - run when an instance is created from an existing one
- `start` - run every time the instance is started

The `template` key points to the template file in the `templates/` directory.

You can pass user-defined template properties to the template file through the `properties` key.

Set the `create_only` key if you want LXD to create the file if it doesn't exist, but not overwrite an existing file.

Template files

Template files use the Pongo2[243] format.

They always receive the following context:

Variable	Type	Description
`trigger`	`string`	Name of the event that triggered the template
`path`	`string`	Path of the file that uses the template
`instance`	`map[string]string`	Key/value map of instance properties (name, architecture, privileged and ephemeral)
`config`	`map[string]string`	Key/value map of the instance's configuration
`devices`	`map[string]map[strir`	Key/value map of the devices assigned to the instance
`properties`	`map[string]string`	Key/value map of the template properties specified in `metadata.yaml`

For convenience, the following functions are exported to the Pongo2 templates:

- `config_get("user.foo", "bar")` - Returns the value of `user.foo`, or `"bar"` if not set.

Image tarballs

LXD supports two LXD-specific image formats: a unified tarball and split tarballs.

These tarballs can be compressed. LXD supports a wide variety of compression algorithms for tarballs. However, for compatibility purposes, you should use `gzip` or `xz`.

Unified tarball

A unified tarball is a single tarball (usually `*.tar.xz`) that contains the full content of the image, including the metadata, the root file system and optionally the template files.

This is the format that LXD itself uses internally when publishing images. It is usually easier to work with; therefore, you should use the unified format when creating LXD-specific images.

The image identifier for such images is the SHA-256 of the tarball.

[243] https://www.schlachter.tech/solutions/pongo2-template-engine/

Split tarballs

A split image consists of two separate tarballs. One tarball contains the metadata and optionally the template files (usually `*.tar.xz`), and the other contains the root file system (usually `*.squashfs` for containers or `*.qcow2` for virtual machines).

For containers, the root file system tarball can be SquashFS-formatted. For virtual machines, the `rootfs.img` file always uses the `qcow2` format. It can optionally be compressed using `qcow2`'s native compression.

This format is designed to allow for easy image building from existing non-LXD rootfs tarballs that are already available. You should also use this format if you want to create images that can be consumed by both LXD and other tools.

The image identifier for such images is the SHA-256 of the concatenation of the metadata and root file system tarball (in that order).

Related topics

How-to guides:

- *Images* (page 148)

Explanation:

- *Local and remote images* (page 348)

4.1.6. Guest OS compatibility

Virtual machines

The following operating systems (OS) were tested as virtual machine guest running on top of on LXD `5.21/stable`. Each OS was tested by doing a manual installation using the official ISO as provided by the vendor.

OS vendor	OS version	OS support	*LXD agent* (page 397)	VirtIO-SCSI	VirtIO-BLK	NVME	CSM (BIOS)	UEFI	Secure Boot
CentOS	CentOS 6.10[1]	EOL	□[2]	□	□[6]	□	□	□	□
CentOS	CentOS 7.9	EOL	□[2]	□	□	□	□	□	□
CentOS	CentOS 8.5	EOL	□	□	□	□	□	□	□
CentOS	CentOS 8-Stream	EOL	□	□	□	□	□	□	□
CentOS	CentOS 9-Stream	Supported	□	□	□	□	□	□	□
Red Hat	RHEL 7.9	EOL	□[2]	□	□	□	□	□	□
Red Hat	RHEL 8.10	Supported	□	□	□	□	□	□	□
Red Hat	RHEL 9.4	Supported	□	□	□	□	□	□	□
SUSE	SLES 12 SP5	Supported	□	□	□	□	□	□	□
SUSE	SLES 15 SP6	Supported	□	□	□	□	□	□	□
Ubuntu	14.04.6 LTS	EOL	□[7]	□	□	□	□	□	□
Ubuntu	16.04.7 LTS	ESM	□[8][9]	□	□	□	□	□	□
Ubuntu	18.04.6 LTS	ESM	□[Page 397, 9]	□	□	□	□	□	□
Ubuntu	20.04.6 LTS	Supported	□	□	□	□	□	□	□
Ubuntu	22.04.4 LTS	Supported	□	□	□	□	□	□	□
Ubuntu	24.04.1 LTS	Supported	□	□	□	□	□	□	□
Windows	Server 2012	Supported	□	□	□	□	□	□	□
Windows	Server 2016	Supported	□	□	□	□[3]	□[5]	□	□
Windows	Server 2019	Supported	□	□	□	□	□[Page 397,	□	□
Windows	Server 2022	Supported	□	□	□	□	□[Page 397,	□	□
Windows	10 22H2	Supported	□	□	□	□	□[Page 397,	□	□
Windows	11 23H2[4]	Supported	□	□	□	□	□	□	□

[1] No network support despite having VirtIO-NET module.

[2] Support for 9P or `virtiofs` not available. Note: CentOS 7 has a `kernel-plus` kernel with 9P support allowing

Legend	Icon
recommended	□
supported	□
not applicable	□
not supported	□

Notes

LXD agent

The LXD agent provides the ability to execute commands inside of the virtual machine guest without relying on traditional access solution like secure shell (SSH) or Remote Desktop Protocol (RDP). This agent is only supported on Linux guests using `systemd`. For how to manually setup the agent, see *Install the LXD agent into virtual machine instances* (page 83).

CSM/BIOS boot

```
lxc config set v1 security.secureboot=false
lxc config set v1 security.csm=true
```

Virtual TPM

```
lxc config device add v1 vtpm tpm path=/dev/tpm0
```

VirtIO-BLK or NVMe

```
lxc config device override v1 root io.bus=virtio-blk
# or
lxc config device override v1 root io.bus=nvme
```

Disconnect the ISO

```
lxc config device remove v1 iso
```

Containers

Unlike virtual machines, container guests rely on the host's kernel for execution. Since each Linux distribution ships with a unique set of features supported by their official kernels, the possibilities are almost endless. As such, the following compatibility table focuses on hosts

LXD agent to work (with `selinux=0`).

[6] The OS installer hangs when booting with VirtIO-BLK despite having VirtIO-BLK supported by the kernel.

[7] This Linux version does not use `systemd` which the LXD agent requires.

[8] Requires the HWE kernel (`4.15`) for proper `vsock` support which is required by the LXD agent.

[9] The `lxd-agent-installer` package is not available so `lxd-agent` has to be manually setup (see *Install the LXD agent into virtual machine instances* (page 83)) or through `cloud-init` (see *VM cloud-init* (page 479)).

[3] NVMe disks are visible but the installer lists all 255 namespaces slowing down the initialization.

[5] The OS installer hangs when booting in CSM/BIOS mode.

[4] A virtual TPM is required.

running Ubuntu LTS releases with LXD `5.21/stable` and Ubuntu releases as container guests. The main compatibility factor is the `cgroup` version required by the container and supported by the host.

Host OS / Guest OS	Ubuntu 16.04 LTS	Ubuntu 18.04 LTS	Ubuntu 20.04 LTS	Ubuntu 22.04 LTS	Ubuntu 24.04 LTS	Ubuntu 24.10
Ubuntu 20.04 LTS 5.4.0[10]	□	□	□	□	□	□[11]
Ubuntu 20.04 LTS 5.15.0 (HWE)	□	□	□	□	□	□[12]
Ubuntu 22.04 LTS 5.15.0	□[13]	□	□	□	□	□
Ubuntu 22.04 LTS 6.8.0 (HWE)	□[13]	□	□	□	□	□
Ubuntu 24.04 LTS 6.8.0	□[13]	□	□	□	□	□

Legend	Icon
recommended	□
supported	□
not applicable	□
not supported	□

4.1.7. Container runtime environment

LXD attempts to present a consistent environment to all containers it runs.

The exact environment will differ slightly based on kernel features and user configuration, but otherwise, it is identical for all containers.

File system

LXD assumes that any image it uses to create a new container comes with at least the following root-level directories:

- `/dev` (empty)
- `/proc` (empty)
- `/sbin/init` (executable)
- `/sys` (empty)

[10] The 5.4.0 kernel is below the minimum required version (see *Requirements* (page 385))
[11] Ubuntu 24.10 and later require `cgroupv2` which is not supported by Ubuntu 20.04 LTS regular kernel.
[12] Requires enabling `cgroupv2` support by booting with `systemd.unified_cgroup_hierarchy=1`
[13] Requires enabling `cgroupv1` support by booting with `systemd.unified_cgroup_hierarchy=0`

Devices

LXD containers have a minimal and ephemeral `/dev` based on a `tmpfs` file system. Since this is a `tmpfs` and not a `devtmpfs` file system, device nodes appear only if manually created.

The following standard set of device nodes is set up automatically:

- `/dev/console`
- `/dev/fd`
- `/dev/full`
- `/dev/log`
- `/dev/null`
- `/dev/ptmx`
- `/dev/random`
- `/dev/stdin`
- `/dev/stderr`
- `/dev/stdout`
- `/dev/tty`
- `/dev/urandom`
- `/dev/zero`

In addition to the standard set of devices, the following devices are also set up for convenience:

- `/dev/fuse`
- `/dev/net/tun`
- `/dev/mqueue`

Network

LXD containers may have any number of network devices attached to them. The naming for those (unless overridden by the user) is `ethX`, where `X` is an incrementing number.

Container-to-host communication

LXD sets up a socket at `/dev/lxd/sock` that the root user in the container can use to communicate with LXD on the host.

See *Communication between instance and host* (page 685) for the API documentation.

Mounts

The following mounts are set up by default:

- `/proc` ()
- `/sys` (`sysfs`)
- `/sys/fs/cgroup/*` (`cgroupfs`) (only on kernels that lack cgroup namespace support)

If they are present on the host, the following paths will also automatically be mounted:

- `/proc/sys/fs/binfmt_misc` (only on kernels that lack `binfmt_misc` namespace support)
- `/sys/firmware/efi/efivars`
- `/sys/fs/fuse/connections`
- `/sys/fs/pstore`
- `/sys/kernel/debug`
- `/sys/kernel/security`

The reason for passing all of those paths is that legacy init systems require them to be mounted, or be mountable, inside the container.

The majority of those paths will not be writable (or even readable) from inside an unprivileged container. In privileged containers, they will be blocked by the AppArmor policy.

LXCFS

If LXCFS is present on the host, it is automatically set up for the container.

This normally results in a number of `/proc` files being overridden through bind-mounts. On older kernels, a virtual version of `/sys/fs/cgroup` might also be set up by LXCFS.

PID1

LXD spawns whatever is located at `/sbin/init` as the initial process of the container (PID 1). This binary should act as a proper init system, including handling re-parented processes.

LXD's communication with PID1 in the container is limited to two signals:

- `SIGINT` to trigger a reboot of the container
- `SIGPWR` (or alternatively `SIGRTMIN+3`) to trigger a clean shutdown of the container

The initial environment of PID1 is blank except for `container=lxc`, which can be used by the init system to detect the runtime.

All file descriptors above the default three are closed prior to PID1 being spawned.

Related topics

How-to guides:

- *Instances* (page 73)

Explanation:

- *Instance types in LXD* (page 347)

4.2. Configuration options

LXD is highly configurable. Check the available configuration options for the LXD server and the different entities used in LXD.

4.2.1. Index

4.2.2. Server configuration

The LXD server can be configured through a set of key/value configuration options.

The key/value configuration is namespaced. The following options are available:

See *How to configure the LXD server* (page 48) for instructions on how to set the configuration options.

> **Note**
>
> Options marked with a `global` scope are immediately applied to all cluster members. Options with a `local` scope must be set on a per-member basis.

Core configuration

The following server options control the core daemon configuration: `core.bgp_address` Address to bind the BGP server to (page 401)

Key:	`core.bgp_address`
Type:	string
Scope:	local

See *How to configure LXD as a BGP server* (page 214).

`core.bgp_asn` BGP Autonomous System Number for the local server (page 401)

Key:	`core.bgp_asn`
Type:	string
Scope:	global

`core.bgp_routerid` A unique identifier for the BGP server (page 401)

Key:	`core.bgp_routerid`
Type:	string
Scope:	local

The identifier must be formatted as an IPv4 address.

`core.debug_address` Address to bind the `pprof`[244] debug server to (HTTP) (page 402)

Key:	`core.debug_address`
Type:	string
Scope:	local

`core.dns_address` Address to bind the authoritative DNS server to (page 402)

Key:	`core.dns_address`
Type:	string
Scope:	local

See *Enable the built-in DNS server* (page 254).

`core.https_address` Address to bind for the remote API (HTTPS) (page 402)

Key:	`core.https_address`
Type:	string
Scope:	local

See *How to expose LXD to the network* (page 44).

`core.https_allowed_credentials` Whether to set `Access-Control-Allow-Credentials` (page 402)

Key:	`core.https_allowed_credentials`
Type:	bool
Default:	`false`
Scope:	global

If enabled, the `Access-Control-Allow-Credentials` HTTP header value is set to `true`.

`core.https_allowed_headers` `Access-Control-Allow-Headers` HTTP header value (page 402)

Key:	`core.https_allowed_headers`
Type:	string
Scope:	global

`core.https_allowed_methods` `Access-Control-Allow-Methods` HTTP header value (page 402)

[244] https://pkg.go.dev/net/http/pprof

Key:	`core.https_allowed_methods`
Type:	string
Scope:	global

`core.https_allowed_origin` `Access-Control-Allow-Origin` HTTP header value (page 403)

Key:	`core.https_allowed_origin`
Type:	string
Scope:	global

`core.https_trusted_proxy` Trusted servers to provide the client's address (page 403)

Key:	`core.https_trusted_proxy`
Type:	string
Scope:	global

Specify a comma-separated list of IP addresses of trusted servers that provide the client's address through the proxy connection header.

`core.metrics_address` Address to bind the metrics server to (HTTPS) (page 403)

Key:	`core.metrics_address`
Type:	string
Scope:	local

See *How to monitor metrics* (page 301).

`core.metrics_authentication` Whether to enforce authentication on the metrics endpoint (page 403)

Key:	`core.metrics_authentication`
Type:	bool
Default:	`true`
Scope:	global

`core.proxy_http` HTTP proxy to use (page 403)

Key:	`core.proxy_http`
Type:	string
Scope:	global

If this option is not specified, LXD falls back to the `HTTP_PROXY` environment variable (if set).

`core.proxy_https` HTTPS proxy to use (page 404)

Key:	`core.proxy_https`
Type:	string
Scope:	global

If this option is not specified, LXD falls back to the `HTTPS_PROXY` environment variable (if set).

`core.proxy_ignore_hosts` Hosts that don't need the proxy (page 404)

Key:	`core.proxy_ignore_hosts`
Type:	string
Scope:	global

Specify this option in a similar format to `NO_PROXY` (for example, `1.2.3.4,1.2.3.5`)

If this option is not specified, LXD falls back to the `NO_PROXY` environment variable (if set).

`core.remote_token_expiry` Time after which a remote add token expires (page 404)

Key:	`core.remote_token_expiry`
Type:	string
Default:	no expiry
Scope:	global

`core.shutdown_timeout` How long to wait before shutdown (page 404)

Key:	`core.shutdown_timeout`
Type:	integer
Default:	`5`
Scope:	global

Specify the number of minutes to wait for running operations to complete before the LXD server shuts down.

`core.storage_buckets_address` Address to bind the storage object server to (HTTPS) (page 404)

Key:	`core.storage_buckets_address`
Type:	string
Scope:	local

See *How to manage storage buckets and keys* (page 193).

core.syslog_socket Whether to enable the syslog unixgram socket listener (page 404)

Key:	core.syslog_socket
Type:	bool
Default:	false
Scope:	local

Set this option to true to enable the syslog unixgram socket to receive log messages from external processes.

core.trust_ca_certificates Whether to automatically trust clients signed by the CA (page 405)

Key:	core.trust_ca_certificates
Type:	bool
Default:	false
Scope:	global

ACME configuration

The following server options control the *ACME* (page 361) configuration: acme.agree_tos Agree to ACME terms of service (page 405)

Key:	acme.agree_tos
Type:	bool
Default:	false
Scope:	global

acme.ca_url URL to the directory resource of the ACME service (page 405)

Key:	acme.ca_url
Type:	string
Default:	https://acme-v02.api.letsencrypt.org/directory
Scope:	global

acme.domain Domain for which the certificate is issued (page 405)

Key:	acme.domain
Type:	string
Scope:	global

acme.email Email address used for the account registration (page 405)

Key:	`acme.email`
Type:	string
Scope:	global

OpenID Connect configuration

The following server options configure external user authentication through *OpenID Connect authentication* (page 361): `oidc.audience` Expected audience value for the application (page 406)

Key:	`oidc.audience`
Type:	string
Scope:	global

This value is required by some providers.

`oidc.client.id` OpenID Connect client ID (page 406)

Key:	`oidc.client.id`
Type:	string
Scope:	global

`oidc.client.secret` OpenID Connect client secret (page 406)

Key:	`oidc.client.secret`
Type:	string
Scope:	global

`oidc.groups.claim` A claim used for mapping identity provider groups to LXD groups. (page 406)

Key:	`oidc.groups.claim`
Type:	string
Scope:	global

Specify a custom token claim to denote groups defined at the identity provider. The contents of this claim can be mapped to LXD groups for managing access control. The value of the claim is expected to be a JSON string array.

`oidc.issuer` OpenID Connect Discovery URL for the provider (page 406)

Key:	`oidc.issuer`
Type:	string
Scope:	global

`oidc.scopes` Space-separated list of OpenID Connect scopes (page 407)

Key:	`oidc.scopes`
Type:	space-delimited string
Scope:	global

A list of OpenID Connect scopes to request from the identity provider. This must include the `openid` and `email` scopes. The remaining optional scopes are `offline_access` and `profile`. If you remove the `offline_access` scope, users might be required to log in more frequently. If you remove the `profile` scope, user information may not be displayed in LXD UI (or in `lxc auth identity` commands). You may add additional scopes if this is required by your identity provider, or if necessary for configuration of *identity provider groups* (page 367).

> **Important**
>
> Setting `oidc.client.secret` may prevent CLI clients from authenticating depending on the Identity Provider policies. Set this key only if required by the Identity Provider.

Cluster configuration

The following server options control *Clustering* (page 280): `cluster.healing_threshold` Threshold when to evacuate an offline cluster member (page 407)

Key:	`cluster.healing_threshold`
Type:	integer
Default:	`0`
Scope:	global

Specify the number of seconds after which an offline cluster member is to be evacuated. To disable evacuating offline members, set this option to `0`.

`cluster.https_address` Address to use for clustering traffic (page 407)

Key:	`cluster.https_address`
Type:	string
Scope:	local

See *Separate REST API and clustering networks* (page 289).

`cluster.images_minimal_replica` Number of cluster members that replicate an image (page 407)

Key:	`cluster.images_minimal_replica`
Type:	integer
Default:	`3`
Scope:	global

Specify the minimal number of cluster members that keep a copy of a particular image. Set this option to `1` for no replication, or to `-1` to replicate images on all members.

`cluster.join_token_expiry` Time after which a cluster join token expires (page 408)

Key:	`cluster.join_token_expiry`
Type:	string
Default:	`3H`
Scope:	global

`cluster.max_standby` Number of database stand-by members (page 408)

Key:	`cluster.max_standby`
Type:	integer
Default:	`2`
Scope:	global

Specify the maximum number of cluster members that are assigned the database stand-by role. This must be a number between `0` and `5`.

`cluster.max_voters` Number of database voter members (page 408)

Key:	`cluster.max_voters`
Type:	integer
Default:	`3`
Scope:	global

Specify the maximum number of cluster members that are assigned the database voter role. This must be an odd number >= `3`.

`cluster.offline_threshold` Threshold when an unresponsive member is considered offline (page 408)

Key:	`cluster.offline_threshold`
Type:	integer
Default:	`20`
Scope:	global

Specify the number of seconds after which an unresponsive member is considered offline.

Images configuration

The following server options configure how to handle *Images* (page 148): `images.auto_update_cached` Whether to automatically update cached images (page 409)

Key:	`images.auto_update_cached`
Type:	bool
Default:	`true`
Scope:	global

`images.auto_update_interval` Interval at which to look for updates to cached images (page 409)

Key:	`images.auto_update_interval`
Type:	integer
Default:	`6`
Scope:	global

Specify the interval in hours. To disable looking for updates to cached images, set this option to `0`.

`images.compression_algorithm` Compression algorithm to use for new images (page 409)

Key:	`images.compression_algorithm`
Type:	string
Default:	`gzip`
Scope:	global

Possible values are `bzip2`, `gzip`, `lzma`, `xz`, or `none`.

`images.default_architecture` Default architecture to use in a mixed-architecture cluster (page 409)

Key:	`images.default_architecture`
Type:	string

`images.remote_cache_expiry` When an unused cached remote image is flushed (page 409)

Key:	`images.remote_cache_expiry`
Type:	integer
Default:	`10`
Scope:	global

Specify the number of days after which the unused cached image expires.

Loki configuration

The following server options configure the external log aggregation system: `loki.api.ca_cert` CA certificate for the Loki server (page 410)

Key:	`loki.api.ca_cert`
Type:	string
Scope:	global

`loki.api.url` URL to the Loki server (page 410)

Key:	`loki.api.url`
Type:	string
Scope:	global

Specify the protocol, name or IP and port. For example `https://loki.example.com:3100`. LXD will automatically add the `/loki/api/v1/push` suffix so there's no need to add it here.

`loki.auth.password` Password used for Loki authentication (page 410)

Key:	`loki.auth.password`
Type:	string
Scope:	global

`loki.auth.username` User name used for Loki authentication (page 410)

Key:	`loki.auth.username`
Type:	string
Scope:	global

`loki.instance` Name to use as the instance field in Loki events. (page 410)

Key:	`loki.instance`
Type:	string
Default:	Local server host name or cluster member name
Scope:	global

This allows replacing the default instance value (server host name) by a more relevant value like a cluster identifier.

`loki.labels` Labels for a Loki log entry (page 410)

Key:	`loki.labels`
Type:	string
Scope:	global

Specify a comma-separated list of values that should be used as labels for a Loki log entry.

`loki.loglevel` Minimum log level to send to the Loki server (page 411)

Key:	`loki.loglevel`
Type:	string
Default:	`info`
Scope:	global

`loki.types` Events to send to the Loki server (page 411)

Key:	`loki.types`
Type:	string
Default:	`lifecycle, logging`
Scope:	global

Specify a comma-separated list of events to send to the Loki server. The events can be any combination of `lifecycle`, `logging`, and `ovn`.

Miscellaneous options

The following server options configure server-specific settings for *Instances* (page 73), MAAS integration, *OVN* (page 587) integration, *Backups* (page 316) and *Storage* (page 175):

`backups.compression_algorithm` Compression algorithm to use for backups (page 411)

Key:	`backups.compression_algorithm`
Type:	string
Default:	`gzip`
Scope:	global

Possible values are `bzip2`, `gzip`, `lzma`, `xz`, or `none`.

`instances.migration.stateful` Whether to set `migration.stateful` to `true` for the instances (page 411)

Key:	`instances.migration.stateful`
Type:	bool
Scope:	global

You can override this setting for relevant instances, either in the instance-specific configuration or through a profile.

`instances.nic.host_name` How to set the host name for a NIC (page 411)

Key:	`instances.nic.host_name`
Type:	string
Default:	`random`
Scope:	global

Possible values are `random` and `mac`.

If set to `random`, use the random host interface name as the host name. If set to `mac`, generate a host name in the form `lxd<mac_address>` (MAC without leading two digits).

`instances.placement.scriptlet` Instance placement scriptlet for automatic instance placement (page 412)

Key:	`instances.placement.scriptlet`
Type:	string
Scope:	global

When using custom automatic instance placement logic, this option stores the scriptlet. See *Instance placement scriptlet* (page 373) for more information.

`maas.api.key` API key to manage MAAS (page 412)

Key:	`maas.api.key`
Type:	string
Scope:	global

`maas.api.url` URL of the MAAS server (page 412)

Key:	`maas.api.url`
Type:	string
Scope:	global

`maas.machine` Name of this LXD host in MAAS (page 412)

Key:	`maas.machine`
Type:	string
Default:	host name
Scope:	local

`network.ovn.ca_cert` OVN SSL certificate authority (page 412)

Key:	`network.ovn.ca_cert`
Type:	string
Default:	Content of `/etc/ovn/ovn-central.crt` if present
Scope:	global

`network.ovn.client_cert` OVN SSL client certificate (page 413)

Key:	`network.ovn.client_cert`
Type:	string
Default:	Content of `/etc/ovn/cert_host` if present
Scope:	global

`network.ovn.client_key` OVN SSL client key (page 413)

Key:	`network.ovn.client_key`
Type:	string
Default:	Content of `/etc/ovn/key_host` if present
Scope:	global

`network.ovn.integration_bridge` OVS integration bridge to use for OVN networks (page 413)

Key:	`network.ovn.integration_bridge`
Type:	string
Default:	`br-int`
Scope:	global

`network.ovn.northbound_connection` OVN northbound database connection string (page 413)

Key:	`network.ovn.northbound_connection`
Type:	string
Default:	`unix:/var/run/ovn/ovnnb_db.sock`
Scope:	global

`storage.backups_volume` Volume to use to store backup tarballs (page 413)

Key:	`storage.backups_volume`
Type:	string
Scope:	local

Specify the volume using the syntax `POOL/VOLUME`.

`storage.images_volume` Volume to use to store the image tarballs (page 413)

Key:	`storage.images_volume`
Type:	string
Scope:	local

Specify the volume using the syntax `POOL/VOLUME`.

Related topics

How-to guides:

- *How to configure the LXD server* (page 48)

4.2.3. Instance configuration

The instance configuration consists of different categories:

Instance properties

Instance properties are specified when the instance is created. They include, for example, the instance name and architecture. Some of the properties are read-only and cannot be changed after creation, while others can be updated by *setting their property value* (page 86) or *editing the full instance configuration* (page 91).

In the YAML configuration, properties are on the top level.

See *Instance properties* (page 415) for a reference of available instance properties.

Instance options

Instance options are configuration options that are related directly to the instance. They include, for example, startup options, security settings, hardware limits, kernel modules, snapshots and user keys. These options can be specified as key/value pairs during instance creation (through the `--config key=value` flag). After creation, they can be configured with the `lxc config set` (page 747) and `lxc config unset` (page 760) commands.

In the YAML configuration, options are located under the `config` entry.

See *Instance options* (page 415) for a reference of available instance options, and *Configure instance options* (page 84) for instructions on how to configure the options.

Instance devices

Instance devices are attached to an instance. They include, for example, network interfaces, mount points, USB and GPU devices. Devices are usually added after an instance is created with the `lxc config device add` (page 739) command, but they can also be added to a profile or a YAML configuration file that is used to create an instance.

Each type of device has its own specific set of options, referred to as *instance device options*.

In the YAML configuration, devices are located under the `devices` entry.

See *Devices* (page 447) for a reference of available devices and the corresponding instance device options, and *Configure devices* (page 87) for instructions on how to add and configure instance devices.

Instance properties

Instance properties are set when the instance is created. They cannot be part of a *profile* (page 97).

The following instance properties are available: `architecture` Instance architecture (page 415)

Key:	`architecture`
Type:	string
Read-only:	no

`name` Instance name (page 415)

Key:	`name`
Type:	string
Read-only:	yes

See *Instance name requirements* (page 415).

Instance name requirements

The instance name can be changed only by renaming the instance with the *`lxc rename`* (page 876) command.

Valid instance names must fulfill the following requirements:

- The name must be between 1 and 63 characters long.
- The name must contain only letters, numbers and dashes from the ASCII table.
- The name must not start with a digit or a dash.
- The name must not end with a dash.

The purpose of these requirements is to ensure that the instance name can be used in DNS records, on the file system, in various security profiles and as the host name of the instance itself.

Instance options

Instance options are configuration options that are directly related to the instance.

See *Configure instance options* (page 84) for instructions on how to set the instance options.

The key/value configuration is namespaced. The following options are available:

- *Miscellaneous options* (page 416)
- *Boot-related options* (page 418)
- *`cloud-init` configuration* (page 419)
- *Resource limits* (page 421)
- *Migration options* (page 428)
- *NVIDIA and CUDA configuration* (page 429)

- *Raw instance configuration overrides* (page 430)
- *Security policies* (page 433)
- *Snapshot scheduling and configuration* (page 441)
- *Volatile internal data* (page 443)

Note that while a type is defined for each option, all values are stored as strings and should be exported over the REST API as strings (which makes it possible to support any extra values without breaking backward compatibility).

Miscellaneous options

In addition to the configuration options listed in the following sections, these instance options are supported: `agent.nic_config` Whether to use the name and MTU of the default network interfaces (page 416)

Key:	`agent.nic_config`
Type:	bool
Default:	`false`
Live update:	no
Condition:	virtual machine

When set to true, the name and MTU of the default network interfaces inside the virtual machine will match those of the instance devices.

`cluster.evacuate` What to do when evacuating the instance (page 416)

Key:	`cluster.evacuate`
Type:	string
Default:	`auto`
Live update:	no

The `cluster.evacuate` provides control over how instances are handled when a cluster member is being evacuated.

Available Modes:

- `auto` *(default)*: The system will automatically decide the best evacuation method based on the instance's type and configured devices:
  - If any device is not suitable for migration, the instance will not be migrated (only stopped).
  - Live migration will be used only for virtual machines with the `migration.stateful` setting enabled and for which all its devices can be migrated as well.
- `live-migrate`: Instances are live-migrated to another node. This means the instance remains running and operational during the migration process, ensuring minimal disruption.

- `migrate`: In this mode, instances are migrated to another node in the cluster. The migration process will not be live, meaning there will be a brief downtime for the instance during the migration.
- `stop`: Instances are not migrated. Instead, they are stopped on the current node.

See *Evacuate a cluster member* (page 286) for more information.

`linux.kernel_modules` Kernel modules to load or allow loading (page 417)

Key:	`linux.kernel_modules`
Type:	string
Live update:	yes
Condition:	container

Specify the kernel modules as a comma-separated list.

The modules are loaded before the instance starts, or they can be loaded by a privileged user if *`linux.kernel_modules.load`* (page 417) is set to `ondemand`.

`linux.kernel_modules.load` How to load kernel modules (page 417)

Key:	`linux.kernel_modules.load`
Type:	string
Default:	`boot`
Live update:	no
Condition:	container

This option specifies how to load the kernel modules that are specified in *`linux.kernel_modules`* (page 417). Possible values are `boot` (load the modules when booting the container) and `ondemand` (intercept the `finit_modules()` syscall and allow a privileged user in the container's user namespace to load the modules).

`linux.sysctl.*` Override for the corresponding `sysctl` setting in the container (page 417)

Key:	`linux.sysctl.*`
Type:	string
Live update:	no
Condition:	container

`ubuntu_pro.guest_attach` Whether to auto-attach Ubuntu Pro. (page 417)

Key:	`ubuntu_pro.guest_attach`
Type:	string
Live update:	no

Indicate whether the guest should auto-attach Ubuntu Pro at start up.

See *How to configure Ubuntu Pro guest attachment* (page 104) for more information.

`user.*` Free-form user key/value storage (page 418)

Key:	`user.*`
Type:	string
Live update:	no

User keys can be used in search.

`environment.*` Environment variables for the instance (page 418)

Key:	`environment.*`
Type:	string
Live update:	yes (exec)

You can export key/value environment variables to the instance. These are then set for *lxc exec* (page 763).

Boot-related options

The following instance options control the boot-related behavior of the instance: `boot.autostart` Whether to always start the instance when LXD starts (page 418)

Key:	`boot.autostart`
Type:	bool
Live update:	no

If set to `false`, restore the last state.

`boot.autostart.delay` Delay after starting the instance (page 418)

Key:	`boot.autostart.delay`
Type:	integer
Default:	`0`
Live update:	no

The number of seconds to wait after the instance started before starting the next one.

`boot.autostart.priority` What order to start the instances in (page 418)

Key:	boot.autostart.priority
Type:	integer
Default:	0
Live update:	no

The instance with the highest value is started first.

boot.debug_edk2 Enable debug version of the edk2 (page 419)

Key:	boot.debug_edk2
Type:	bool

The instance should use a debug version of the edk2. A log file can be found in $LXD_DIR/logs/<instance_name>/edk2.log.

boot.host_shutdown_timeout How long to wait for the instance to shut down (page 419)

Key:	boot.host_shutdown_timeout
Type:	integer
Default:	30
Live update:	yes

Number of seconds to wait for the instance to shut down before it is force-stopped.

boot.stop.priority What order to shut down the instances in (page 419)

Key:	boot.stop.priority
Type:	integer
Default:	0
Live update:	no

The instance with the highest value is shut down first.

cloud-init configuration

The following instance options control the *cloud-init* (page 116) configuration of the instance: cloud-init.network-config Network configuration for cloud-init (page 419)

Key:	cloud-init.network-config
Type:	string
Default:	DHCP on eth0
Live update:	no
Condition:	If supported by image

The content is used as seed value for `cloud-init`.

`cloud-init.ssh-keys.KEYNAME` Additional SSH key to be injected on the instance by `cloud-init` (page 420)

Key:	`cloud-init.ssh-keys.KEYNAME`
Type:	string
Live update:	no
Condition:	If supported by image

Represents an additional SSH public key to be merged into existing `cloud-init` seed data and injected into an instance. Has the format `{user}:{key}`, where {user} is a Linux username and {key} can be either a pure SSH public key or an import ID for a key hosted elsewhere. // For example: `root:gh:githubUser`, `myUser:ssh-keyAlg publicKeyHash`

`cloud-init.user-data` User data for `cloud-init` (page 420)

Key:	`cloud-init.user-data`
Type:	string
Default:	`#cloud-config`
Live update:	no
Condition:	If supported by image

The content is used as seed value for `cloud-init`.

`cloud-init.vendor-data` Vendor data for `cloud-init` (page 420)

Key:	`cloud-init.vendor-data`
Type:	string
Default:	`#cloud-config`
Live update:	no
Condition:	If supported by image

The content is used as seed value for `cloud-init`.

`user.network-config` Legacy version of `cloud-init.network-config` (page 420)

Key:	`user.network-config`
Type:	string
Default:	`DHCP on eth0`
Live update:	no
Condition:	If supported by image

`user.user-data` Legacy version of `cloud-init.user-data` (page 420)

Key:	`user.user-data`
Type:	string
Default:	`#cloud-config`
Live update:	no
Condition:	If supported by image

`user.vendor-data` Legacy version of `cloud-init.vendor-data` (page 421)

Key:	`user.vendor-data`
Type:	string
Default:	`#cloud-config`
Live update:	no
Condition:	If supported by image

Support for these options depends on the image that is used and is not guaranteed.

If you specify both `cloud-init.user-data` and `cloud-init.vendor-data`, the content of both options is merged. Therefore, make sure that the `cloud-init` configuration you specify in those options does not contain the same keys.

Resource limits

The following instance options specify resource limits for the instance: `limits.cpu` Which CPUs to expose to the instance (page 421)

Key:	`limits.cpu`
Type:	string
Default:	1 (VMs)
Live update:	yes

A number or a specific range of CPUs to expose to the instance.

See *CPU pinning* (page 426) for more information.

`limits.cpu.allowance` How much of the CPU can be used (page 421)

Key:	`limits.cpu.allowance`
Type:	string
Default:	100%
Live update:	yes
Condition:	container

To control how much of the CPU can be used, specify either a percentage (`50%`) for a soft limit or a chunk of time (`25ms/100ms`) for a hard limit.

See *Allowance and priority (container only)* (page 427) for more information.

`limits.cpu.nodes` Which NUMA nodes to place the instance CPUs on (page 421)

Key:	`limits.cpu.nodes`
Type:	string
Live update:	yes

A comma-separated list of NUMA node IDs or ranges to place the instance CPUs on.

See *Allowance and priority (container only)* (page 427) for more information.

`limits.cpu.pin_strategy` VM CPU auto pinning strategy (page 422)

Key:	`limits.cpu.pin_strategy`
Type:	string
Default:	`none`
Live update:	no
Condition:	virtual machine

Specify the strategy for VM CPU auto pinning. Possible values: `none` (disables CPU auto pinning) and `auto` (enables CPU auto pinning).

See *CPU limits for virtual machines* (page 426) for more information.

`limits.cpu.priority` CPU scheduling priority compared to other instances (page 422)

Key:	`limits.cpu.priority`
Type:	integer
Default:	`10` (maximum)
Live update:	yes
Condition:	container

When overcommitting resources, specify the CPU scheduling priority compared to other instances that share the same CPUs. Specify an integer between 0 and 10.

See *Allowance and priority (container only)* (page 427) for more information.

`limits.disk.priority` Priority of the instance's I/O requests (page 422)

Key:	`limits.disk.priority`
Type:	integer
Default:	`5` (medium)
Live update:	yes

Controls how much priority to give to the instance's I/O requests when under load.

Specify an integer between 0 and 10.

`limits.hugepages.1GB` Limit for the number of 1 GB huge pages (page 422)

Key:	`limits.hugepages.1GB`
Type:	string
Live update:	yes
Condition:	container

Fixed value (in bytes) to limit the number of 1 GB huge pages. Various suffixes are supported (see *Units for storage and network limits* (page 506)).

See *Huge page limits* (page 427) for more information.

`limits.hugepages.1MB` Limit for the number of 1 MB huge pages (page 423)

Key:	`limits.hugepages.1MB`
Type:	string
Live update:	yes
Condition:	container

Fixed value (in bytes) to limit the number of 1 MB huge pages. Various suffixes are supported (see *Units for storage and network limits* (page 506)).

See *Huge page limits* (page 427) for more information.

`limits.hugepages.2MB` Limit for the number of 2 MB huge pages (page 423)

Key:	`limits.hugepages.2MB`
Type:	string
Live update:	yes
Condition:	container

Fixed value (in bytes) to limit the number of 2 MB huge pages. Various suffixes are supported (see *Units for storage and network limits* (page 506)).

See *Huge page limits* (page 427) for more information.

`limits.hugepages.64KB` Limit for the number of 64 KB huge pages (page 423)

Key:	`limits.hugepages.64KB`
Type:	string
Live update:	yes
Condition:	container

Fixed value (in bytes) to limit the number of 64 KB huge pages. Various suffixes are supported (see *Units for storage and network limits* (page 506)).

See *Huge page limits* (page 427) for more information.

`limits.memory` Usage limit for the host's memory (page 423)

Key:	`limits.memory`
Type:	string
Default:	`1GiB` (VMs)
Live update:	yes

Percentage of the host's memory or a fixed value in bytes. Various suffixes are supported.

See *Units for storage and network limits* (page 506) for details.

`limits.memory.enforce` Whether the memory limit is `hard` or `soft` (page 424)

Key:	`limits.memory.enforce`
Type:	string
Default:	`hard`
Live update:	yes
Condition:	container

If the instance's memory limit is `hard`, the instance cannot exceed its limit. If it is `soft`, the instance can exceed its memory limit when extra host memory is available.

`limits.memory.hugepages` Whether to back the instance using huge pages (page 424)

Key:	`limits.memory.hugepages`
Type:	bool
Default:	`false`
Live update:	no
Condition:	virtual machine

If this option is set to `false`, regular system memory is used.

`limits.memory.swap` Whether to encourage/discourage swapping less used pages for this instance (page 424)

Key:	`limits.memory.swap`
Type:	bool
Default:	`true`
Live update:	yes
Condition:	container

`limits.memory.swap.priority` Prevents the instance from being swapped to disk (page 424)

Key:	`limits.memory.swap.priority`
Type:	integer
Default:	`10` (maximum)
Live update:	yes
Condition:	container

Specify an integer between 0 and 10. The higher the value, the less likely the instance is to be swapped to disk.

`limits.processes` Maximum number of processes that can run in the instance (page 425)

Key:	`limits.processes`
Type:	integer
Default:	empty
Live update:	yes
Condition:	container

If left empty, no limit is set.

`limits.kernel.*` Kernel resources per instance (page 425)

Key:	`limits.kernel.*`
Type:	string
Live update:	no
Condition:	container

You can set kernel limits on an instance, for example, you can limit the number of open files. See *Kernel resource limits* (page 427) for more information.

CPU limits

You have different options to limit CPU usage:

- Set *`limits.cpu`* (page 421) to restrict which CPUs the instance can see and use. See *CPU pinning* (page 426) for how to set this option.
- Set *`limits.cpu.allowance`* (page 421) to restrict the load an instance can put on the available CPUs. This option is available only for containers. See *Allowance and priority (container only)* (page 427) for how to set this option.
- Set *`limits.cpu.pin_strategy`* (page 422) to specify the strategy for virtual-machine CPU auto pinning. This option is available only for virtual machines. See *CPU limits for virtual machines* (page 426) for how to set this option.

It is possible to set both options at the same time to restrict both which CPUs are visible to the instance and the allowed usage of those instances. However, if you use *`limits.cpu.allowance`* (page 421) with a time limit, you should avoid using *`limits.cpu`* (page 421) in addition, because that puts a lot of constraints on the scheduler and might lead to less efficient allocations.

The CPU limits are implemented through a mix of the `cpuset` and `cpu` cgroup controllers.

CPU pinning

limits.cpu (page 421) results in CPU pinning through the `cpuset` controller. You can specify either which CPUs or how many CPUs are visible and available to the instance:

- To specify which CPUs to use, set `limits.cpu` to either a set of CPUs (for example, `1,2,3`) or a CPU range (for example, `0-3`).

  To pin to a single CPU, use the range syntax (for example, `1-1`) to differentiate it from a number of CPUs.
- If you specify a number (for example, `4`) of CPUs, LXD will do dynamic load-balancing of all instances that aren't pinned to specific CPUs, trying to spread the load on the machine. Instances are re-balanced every time an instance starts or stops, as well as whenever a CPU is added to the system.

CPU limits for virtual machines

Note

LXD supports live-updating the *limits.cpu* (page 421) option. However, for virtual machines, this only means that the respective CPUs are hotplugged. Depending on the guest operating system, you might need to either restart the instance or complete some manual actions to bring the new CPUs online.

LXD virtual machines default to having just one vCPU allocated, which shows up as matching the host CPU vendor and type, but has a single core and no threads.

When *limits.cpu* (page 421) is set to a single integer, LXD allocates multiple vCPUs and exposes them to the guest as full cores. Unless *limits.cpu.pin_strategy* (page 422) is set to `auto`, those vCPUs are not pinned to specific cores on the host. The number of vCPUs can be updated while the VM is running.

When *limits.cpu* (page 421) is set to a range or comma-separated list of CPU IDs (as provided by *lxc info --resources* (page 782)), the vCPUs are pinned to those cores. In this scenario, LXD checks whether the CPU configuration lines up with a realistic hardware topology and if it does, it replicates that topology in the guest. When doing CPU pinning, it is not possible to change the configuration while the VM is running.

For example, if the pinning configuration includes eight threads, with each pair of thread coming from the same core and an even number of cores spread across two CPUs, the guest will show two CPUs, each with two cores and each core with two threads. The NUMA layout is similarly replicated and in this scenario, the guest would most likely end up with two NUMA nodes, one for each CPU socket.

In such an environment with multiple NUMA nodes, the memory is similarly divided across NUMA nodes and be pinned accordingly on the host and then exposed to the guest.

All this allows for very high performance operations in the guest as the guest scheduler can properly reason about sockets, cores and threads as well as consider NUMA topology when sharing memory or moving processes across NUMA nodes.

Allowance and priority (container only)

limits.cpu.allowance (page 421) drives either the CFS scheduler quotas when passed a time constraint, or the generic CPU shares mechanism when passed a percentage value:

- The time constraint (for example, `20ms/50ms`) is a hard limit. For example, if you want to allow the container to use a maximum of one CPU, set *limits.cpu.allowance* (page 421) to a value like `100ms/100ms`. The value is relative to one CPU worth of time, so to restrict to two CPUs worth of time, use something like `100ms/50ms` or `200ms/100ms`.
- When using a percentage value, the limit is a soft limit that is applied only when under load. It is used to calculate the scheduler priority for the instance, relative to any other instance that is using the same CPU or CPUs. For example, to limit the CPU usage of the container to one CPU when under load, set *limits.cpu.allowance* (page 421) to `100%`.

limits.cpu.nodes (page 421) can be used to restrict the CPUs that the instance can use to a specific set of NUMA nodes. To specify which NUMA nodes to use, set *limits.cpu.nodes* (page 421) to either a set of NUMA node IDs (for example, `0,1`) or a set of NUMA node ranges (for example, `0-1,2-4`).

limits.cpu.priority (page 422) is another factor that is used to compute the scheduler priority score when a number of instances sharing a set of CPUs have the same percentage of CPU assigned to them.

Huge page limits

LXD allows to limit the number of huge pages available to a container through the `limits.hugepage.[size]` key (for example, *limits.hugepages.1MB* (page 423)).

Architectures often expose multiple huge-page sizes. The available huge-page sizes depend on the architecture.

Setting limits for huge pages is especially useful when LXD is configured to intercept the `mount` syscall for the `hugetlbfs` file system in unprivileged containers. When LXD intercepts a `hugetlbfs` `mount` syscall, it mounts the `hugetlbfs` file system for a container with correct `uid` and `gid` values as mount options. This makes it possible to use huge pages from unprivileged containers. However, it is recommended to limit the number of huge pages available to the container through `limits.hugepages.[size]` to stop the container from being able to exhaust the huge pages available to the host.

Limiting huge pages is done through the `hugetlb` cgroup controller, which means that the host system must expose the `hugetlb` controller in the legacy or unified cgroup hierarchy for these limits to apply.

Kernel resource limits

For container instances, LXD exposes a generic namespaced key *limits.kernel.** (page 425) that can be used to set resource limits.

It is generic in the sense that LXD does not perform any validation on the resource that is specified following the `limits.kernel.*` prefix. LXD cannot know about all the possible resources that a given kernel supports. Instead, LXD simply passes down the corresponding resource key after the `limits.kernel.*` prefix and its value to the kernel. The kernel does the appropriate validation. This allows users to specify any supported limit on their system.

Some common limits are:

Key	Resource	Description
`limits.kernel.as`	`RLIMIT_AS`	Maximum size of the process's virtual memory
`limits.kernel.core`	`RLIMIT_CORE`	Maximum size of the process's core dump file
`limits.kernel.cpu`	`RLIMIT_CPU`	Limit in seconds on the amount of CPU time the process can consume
`limits.kernel.data`	`RLIMIT_DATA`	Maximum size of the process's data segment
`limits.kernel.fsize`	`RLIMIT_FSIZE`	Maximum size of files the process may create
`limits.kernel.locks`	`RLIMIT_LOCKS`	Limit on the number of file locks that this process may establish
`limits.kernel.memlock`	`RLIMIT_MEMLO`	Limit on the number of bytes of memory that the process may lock in RAM
`limits.kernel.nice`	`RLIMIT_NICE`	Maximum value to which the process's nice value can be raised
`limits.kernel.nofile`	`RLIMIT_NOFIL`	Maximum number of open files for the process
`limits.kernel.nproc`	`RLIMIT_NPROC`	Maximum number of processes that can be created for the user of the calling process
`limits.kernel.rtprio`	`RLIMIT_RTPRI`	Maximum value on the real-time-priority that may be set for this process
`limits.kernel.sigpending`	`RLIMIT_SIGPE`	Maximum number of signals that may be queued for the user of the calling process

A full list of all available limits can be found in the manpages for the `getrlimit(2)`/`setrlimit(2)` system calls.

To specify a limit within the `limits.kernel.*` namespace, use the resource name in lowercase without the `RLIMIT_` prefix. For example, `RLIMIT_NOFILE` should be specified as `nofile`.

A limit is specified as two colon-separated values that are either numeric or the word `unlimited` (for example, `limits.kernel.nofile=1000:2000`). A single value can be used as a shortcut to set both soft and hard limit to the same value (for example, `limits.kernel.nofile=3000`).

A resource with no explicitly configured limit will inherit its limit from the process that starts up the container. Note that this inheritance is not enforced by LXD but by the kernel.

Migration options

The following instance options control the behavior if the instance is *moved from one LXD server to another* (page 135): `migration.incremental.memory` Whether to use incremental memory transfer (page 428)

Key:	`migration.incremental.memory`
Type:	bool
Default:	`false`
Live update:	yes
Condition:	container

Using incremental memory transfer of the instance's memory can reduce downtime.

`migration.incremental.memory.goal` Percentage of memory to have in sync before stopping the instance (page 429)

Key:	`migration.incremental.memory.goal`
Type:	integer
Default:	`70`
Live update:	yes
Condition:	container

`migration.incremental.memory.iterations` Maximum number of transfer operations to go through before stopping the instance (page 429)

Key:	`migration.incremental.memory.iterations`
Type:	integer
Default:	`10`
Live update:	yes
Condition:	container

`migration.stateful` Whether to allow for stateful stop/start and snapshots (page 429)

Key:	`migration.stateful`
Type:	bool
Default:	`false` or value from profiles or `instances.migration.stateful` (if set)
Live update:	no
Condition:	virtual machine

Enabling this option prevents the use of some features that are incompatible with it.

NVIDIA and CUDA configuration

The following instance options specify the NVIDIA and CUDA configuration of the instance:
`nvidia.driver.capabilities` What driver capabilities the instance needs (page 429)

Key:	`nvidia.driver.capabilities`
Type:	string
Default:	`compute,utility`
Live update:	no
Condition:	container

The specified driver capabilities are used to set `libnvidia-container` `NVIDIA_DRIVER_CAPABILITIES`.

`nvidia.require.cuda` Required CUDA version (page 430)

Key:	`nvidia.require.cuda`
Type:	string
Live update:	no
Condition:	container

The specified version expression is used to set `libnvidia-container` `NVIDIA_REQUIRE_CUDA`.

`nvidia.require.driver` Required driver version (page 430)

Key:	`nvidia.require.driver`
Type:	string
Live update:	no
Condition:	container

The specified version expression is used to set `libnvidia-container` `NVIDIA_REQUIRE_DRIVER`.

`nvidia.runtime` Whether to pass the host NVIDIA and CUDA runtime libraries into the instance (page 430)

Key:	`nvidia.runtime`
Type:	bool
Default:	`false`
Live update:	no
Condition:	container

Raw instance configuration overrides

The following instance options allow direct interaction with the backend features that LXD itself uses: `raw.apparmor` AppArmor profile entries (page 430)

Key:	`raw.apparmor`
Type:	blob
Live update:	yes

The specified entries are appended to the generated profile.

`raw.idmap` Raw idmap configuration (page 431)

Key:	`raw.idmap`
Type:	blob
Live update:	no
Condition:	unprivileged container

For example: `both 1000 1000`

`raw.lxc` Raw LXC configuration to be appended to the generated one (page 431)

Key:	`raw.lxc`
Type:	blob
Live update:	no
Condition:	container

`raw.qemu` Raw QEMU configuration to be appended to the generated command line (page 431)

Key:	`raw.qemu`
Type:	blob
Live update:	no
Condition:	virtual machine

`raw.qemu.conf` Addition/override to the generated `qemu.conf` file (page 431)

Key:	`raw.qemu.conf`
Type:	blob
Live update:	no
Condition:	virtual machine

See *Override QEMU configuration* (page 432) for more information.

`raw.seccomp` Raw Seccomp configuration (page 431)

Key:	`raw.seccomp`
Type:	blob
Live update:	no
Condition:	container

Important

Setting these `raw.*` keys might break LXD in non-obvious ways. Therefore, you should avoid setting any of these keys.

Override QEMU configuration

For VM instances, LXD configures QEMU through a configuration file that is passed to QEMU with the `-readconfig` command-line option. This configuration file is generated for each instance before boot. It can be found at `/var/log/lxd/<instance_name>/qemu.conf`.

The default configuration works fine for LXD's most common use case: modern UEFI guests with VirtIO devices. In some situations, however, you might need to override the generated configuration. For example:

- To run an old guest OS that doesn't support UEFI.
- To specify custom virtual devices when VirtIO is not supported by the guest OS.
- To add devices that are not supported by LXD before the machines boots.
- To remove devices that conflict with the guest OS.

To override the configuration, set the *`raw.qemu.conf`* (page 431) option. It supports a format similar to `qemu.conf`, with some additions. Since it is a multi-line configuration option, you can use it to modify multiple sections or keys.

- To replace a section or key in the generated configuration file, add a section with a different value.

  For example, use the following section to override the default `virtio-gpu-pci` GPU driver:

  ```
  raw.qemu.conf: |-
      [device "qemu_gpu"]
      driver = "qxl-vga"
  ```

- To remove a section, specify a section without any keys. For example:

  ```
  raw.qemu.conf: |-
      [device "qemu_gpu"]
  ```

- To remove a key, specify an empty string as the value. For example:

  ```
  raw.qemu.conf: |-
      [device "qemu_gpu"]
      driver = ""
  ```

- To add a new section, specify a section name that is not present in the configuration file.

The configuration file format used by QEMU allows multiple sections with the same name. Here's a piece of the configuration generated by LXD:

```
[global]
driver = "ICH9-LPC"
property = "disable_s3"
value = "1"

[global]
driver = "ICH9-LPC"
property = "disable_s4"
value = "1"
```

To specify which section to override, specify an index. For example:

```
raw.qemu.conf: |-
    [global][1]
    value = "0"
```

Section indexes start at 0 (which is the default value when not specified), so the above example would generate the following configuration:

```
[global]
driver = "ICH9-LPC"
property = "disable_s3"
value = "1"

[global]
driver = "ICH9-LPC"
property = "disable_s4"
value = "0"
```

Security policies

The following instance options control the *Security* (page 376) policies of the instance:

`security.agent.metrics` Whether the `lxd-agent` is queried for state information and metrics (page 433)

Key:	`security.agent.metrics`
Type:	bool
Default:	`true`
Live update:	no
Condition:	virtual machine

`security.csm` Whether to use a firmware that supports UEFI-incompatible operating systems (page 433)

Key:	`security.csm`
Type:	bool
Default:	`false`
Live update:	no
Condition:	virtual machine

When enabling this option, set *security.secureboot* (page 437) to false.

security.delegate_bpf Whether to enable eBPF delegation using BPF Token mechanism (page 434)

Key:	security.delegate_bpf
Type:	bool
Default:	false
Live update:	no
Condition:	unprivileged container

This option enables BPF functionality delegation mechanism (using BPF Token).

Note: security.delegate_bpf.cmd_types, security.delegate_bpf.map_types, security.delegate_bpf.prog_types, security.delegate_bpf.attach_types need to be configured depending on BPF workload in the container.

See *Privilege delegation using BPF Token* (page 381) for more information.

security.delegate_bpf.attach_types Which eBPF attach types to allow with delegation mechanism (page 434)

Key:	security.delegate_bpf.attach_types
Type:	bool
Default:	false
Live update:	no
Condition:	unprivileged container

Which eBPF program attachment types to allow with delegation mechanism. Syntax follows a kernel one for delegate_attachs bpffs mount option. A number (bitmask) or :-separated list of attachment types to allow can be specified. For example, cgroup_inet_ingress allows BPF_CGROUP_INET_INGRESS attachment type.

security.delegate_bpf.cmd_types Which eBPF commands to allow with delegation mechanism (page 434)

Key:	security.delegate_bpf.cmd_types
Type:	bool
Default:	false
Live update:	no
Condition:	unprivileged container

Which eBPF commands to allow with delegation mechanism. Syntax follows a kernel one for delegate_cmds bpffs mount option. A number (bitmask) or :-separated list of commands to allow can be specified. For example, prog_load:map_create allows eBPF programs loading and eBPF maps creation. Notice: security.delegate_bpf.prog_types and security.delegate_bpf.map_types still need to be configured accordingly.

security.delegate_bpf.map_types Which eBPF maps to allow with delegation mechanism (page 434)

Key:	`security.delegate_bpf.map_types`
Type:	bool
Default:	`false`
Live update:	no
Condition:	unprivileged container

Which eBPF maps to allow with delegation mechanism. Syntax follows a kernel one for `delegate_maps` bpffs mount option. A number (bitmask) or `:`-separated list of map types to allow can be specified. For example, `ringbuf` allows `BPF_MAP_TYPE_RINGBUF` map.

`security.delegate_bpf.prog_types` Which eBPF program types to allow with delegation mechanism (page 435)

Key:	`security.delegate_bpf.prog_types`
Type:	bool
Default:	`false`
Live update:	no
Condition:	unprivileged container

Which eBPF program types to allow with delegation mechanism. Syntax follows a kernel one for `delegate_progs` bpffs mount option. A number (bitmask) or `:`-separated list of program types to allow can be specified. For example, `socket_filter` allows `BPF_PROG_TYPE_SOCKET_FILTER` program type.

`security.devlxd` Whether `/dev/lxd` is present in the instance (page 435)

Key:	`security.devlxd`
Type:	bool
Default:	`true`
Live update:	no

See *Communication between instance and host* (page 685) for more information.

`security.devlxd.images` Controls the availability of the `/1.0/images` API over `devlxd` (page 435)

Key:	`security.devlxd.images`
Type:	bool
Default:	`false`
Live update:	yes

`security.idmap.base` The base host ID to use for the allocation (page 435)

Key:	`security.idmap.base`
Type:	integer
Live update:	no
Condition:	unprivileged container

Setting this option overrides auto-detection.

`security.idmap.isolated` Whether to use a unique idmap for this instance (page 436)

Key:	`security.idmap.isolated`
Type:	bool
Default:	`false`
Live update:	no
Condition:	unprivileged container

If specified, the idmap used for this instance is unique among instances that have this option set.

`security.idmap.size` The size of the idmap to use (page 436)

Key:	`security.idmap.size`
Type:	integer
Live update:	no
Condition:	unprivileged container

`security.nesting` Whether to support running LXD (nested) inside the instance (page 436)

Key:	`security.nesting`
Type:	bool
Default:	`false`
Live update:	yes
Condition:	container

`security.privileged` Whether to run the instance in privileged mode (page 436)

Key:	`security.privileged`
Type:	bool
Default:	`false`
Live update:	no
Condition:	container

See *Container security* (page 378) for more information.

`security.protection.delete` Whether to prevent the instance from being deleted (page 436)

Key:	`security.protection.delete`
Type:	bool
Default:	`false`
Live update:	container

`security.protection.shift` Whether to protect the file system from being UID/GID shifted (page 437)

Key:	`security.protection.shift`
Type:	bool
Default:	`false`
Live update:	yes
Condition:	container

Set this option to `true` to prevent the instance's file system from being UID/GID shifted on startup.

`security.protection.start` Whether to prevent the instance from being started (page 437)

Key:	`security.protection.start`
Type:	bool
Default:	`false`
Live update:	container

`security.secureboot` Whether UEFI secure boot is enabled with the default Microsoft keys (page 437)

Key:	`security.secureboot`
Type:	bool
Default:	`true`
Live update:	no
Condition:	virtual machine

When disabling this option, consider enabling *`security.csm`* (page 433).

`security.sev` Whether AMD SEV (Secure Encrypted Virtualization) is enabled for this VM (page 437)

Key:	`security.sev`
Type:	bool
Default:	`false`
Live update:	no
Condition:	virtual machine

security.sev.policy.es Whether AMD SEV-ES (SEV Encrypted State) is enabled for this VM (page 437)

Key:	security.sev.policy.es
Type:	bool
Default:	false
Live update:	no
Condition:	virtual machine

security.sev.session.data The guest owner's base64-encoded session blob (page 438)

Key:	security.sev.session.data
Type:	string
Default:	true
Live update:	no
Condition:	virtual machine

security.sev.session.dh The guest owner's base64-encoded Diffie-Hellman key (page 438)

Key:	security.sev.session.dh
Type:	string
Default:	true
Live update:	no
Condition:	virtual machine

security.syscalls.allow List of syscalls to allow (page 438)

Key:	security.syscalls.allow
Type:	string
Live update:	no
Condition:	container

A \n-separated list of syscalls to allow. This list must be mutually exclusive with security.syscalls.deny*.

security.syscalls.deny List of syscalls to deny (page 438)

Key:	security.syscalls.deny
Type:	string
Live update:	no
Condition:	container

A \n-separated list of syscalls to deny. This list must be mutually exclusive with security.syscalls.allow.

security.syscalls.deny_compat Whether to block compat_* syscalls (x86_64 only) (page 439)

Key:	security.syscalls.deny_compat
Type:	bool
Default:	false
Live update:	no
Condition:	container

On x86_64, this option controls whether to block compat_* syscalls. On other architectures, the option is ignored.

security.syscalls.deny_default Whether to enable the default syscall deny (page 439)

Key:	security.syscalls.deny_default
Type:	bool
Default:	true
Live update:	no
Condition:	container

security.syscalls.intercept.bpf Whether to handle the bpf() system call (page 439)

Key:	security.syscalls.intercept.bpf
Type:	bool
Default:	false
Live update:	no
Condition:	container

security.syscalls.intercept.bpf.devices Whether to allow BPF programs (page 439)

Key:	security.syscalls.intercept.bpf.devices
Type:	bool
Default:	false
Live update:	no
Condition:	container

This option controls whether to allow BPF programs for the devices cgroup in the unified hierarchy to be loaded.

security.syscalls.intercept.mknod Whether to handle the mknod and mknodat system calls (page 439)

Key:	`security.syscalls.intercept.mknod`
Type:	bool
Default:	`false`
Live update:	no
Condition:	container

These system calls allow creation of a limited subset of char/block devices.

`security.syscalls.intercept.mount` Whether to handle the `mount` system call (page 440)

Key:	`security.syscalls.intercept.mount`
Type:	bool
Default:	`false`
Live update:	no
Condition:	container

`security.syscalls.intercept.mount.allowed` File systems that can be mounted (page 440)

Key:	`security.syscalls.intercept.mount.allowed`
Type:	string
Live update:	yes
Condition:	container

Specify a comma-separated list of file systems that are safe to mount for processes inside the instance.

`security.syscalls.intercept.mount.fuse` File system that should be redirected to FUSE implementation (page 440)

Key:	`security.syscalls.intercept.mount.fuse`
Type:	string
Live update:	yes
Condition:	container

Specify the mounts of a given file system that should be redirected to their FUSE implementation (for example, `ext4=fuse2fs`).

`security.syscalls.intercept.mount.shift` Whether to use idmapped mounts for syscall interception (page 440)

Key:	`security.syscalls.intercept.mount.shift`
Type:	bool
Default:	`false`
Live update:	yes
Condition:	container

`security.syscalls.intercept.sched_setscheduler` Whether to handle the `sched_setscheduler` system call (page 441)

Key:	`security.syscalls.intercept.sched_setscheduler`
Type:	bool
Default:	`false`
Live update:	no
Condition:	container

This system call allows increasing process priority.

`security.syscalls.intercept.setxattr` Whether to handle the `setxattr` system call (page 441)

Key:	`security.syscalls.intercept.setxattr`
Type:	bool
Default:	`false`
Live update:	no
Condition:	container

This system call allows setting a limited subset of restricted extended attributes.

`security.syscalls.intercept.sysinfo` Whether to handle the `sysinfo` system call (page 441)

Key:	`security.syscalls.intercept.sysinfo`
Type:	bool
Default:	`false`
Live update:	no
Condition:	container

This system call can be used to get cgroup-based resource usage information.

Snapshot scheduling and configuration

The following instance options control the creation and expiry of *instance snapshots* (page 128): `snapshots.expiry` When snapshots are to be deleted (page 441)

Key:	`snapshots.expiry`
Type:	string
Live update:	no

Specify an expression like `1M 2H 3d 4w 5m 6y`.

`snapshots.pattern` Template for the snapshot name (page 442)

Key:	`snapshots.pattern`
Type:	string
Default:	`snap%d`
Live update:	no

Specify a Pongo2 template string that represents the snapshot name. This template is used for scheduled snapshots and for unnamed snapshots.

See *Automatic snapshot names* (page 442) for more information.

`snapshots.schedule` Schedule for automatic instance snapshots (page 442)

Key:	`snapshots.schedule`
Type:	string
Default:	empty
Live update:	no

Specify either a cron expression (`<minute> <hour> <dom> <month> <dow>`), a comma-separated list of schedule aliases (`@hourly`, `@daily`, `@midnight`, `@weekly`, `@monthly`, `@annually`, `@yearly`), or leave empty to disable automatic snapshots.

`snapshots.schedule.stopped` Whether to automatically snapshot stopped instances (page 442)

Key:	`snapshots.schedule.stopped`
Type:	bool
Default:	`false`
Live update:	no

Automatic snapshot names

The `snapshots.pattern` option takes a Pongo2 template string to format the snapshot name.

To add a time stamp to the snapshot name, use the Pongo2 context variable `creation_date`. Make sure to format the date in your template string to avoid forbidden characters in the snapshot name. For example, set `snapshots.pattern` to `{{ creation_date|date:'2006-01-02_15-04-05' }}` to name the snapshots after their time of creation, down to the precision of a second.

Another way to avoid name collisions is to use the placeholder %d in the pattern. For the first snapshot, the placeholder is replaced with 0. For subsequent snapshots, the existing snapshot names are taken into account to find the highest number at the placeholder's position. This number is then incremented by one for the new name.

Volatile internal data

> **Warning**
>
> The volatile.* keys cannot be manipulated by the user. Do not attempt to modify these keys in any way. LXD modifies these keys, and attempting to manipulate them yourself might break LXD in non-obvious ways.

The following volatile keys are currently used internally by LXD to store internal data specific to an instance: `volatile.<name>.apply_quota` Disk quota (page 443)

Key:	`volatile.<name>.apply_quota`
Type:	string

The disk quota is applied the next time the instance starts.

`volatile.<name>.ceph_rbd` RBD device path for Ceph disk devices (page 443)

Key:	`volatile.<name>.ceph_rbd`
Type:	string

`volatile.<name>.host_name` Network device name on the host (page 443)

Key:	`volatile.<name>.host_name`
Type:	string

`volatile.<name>.hwaddr` Network device MAC address (page 443)

Key:	`volatile.<name>.hwaddr`
Type:	string

The network device MAC address is used when no `hwaddr` property is set on the device itself.

`volatile.<name>.last_state.created` Whether the network device physical device was created (page 443)

Key:	volatile.<name>.last_state.created
Type:	string

Possible values are true or false.

volatile.<name>.last_state.hwaddr Network device original MAC (page 444)

Key:	volatile.<name>.last_state.hwaddr
Type:	string

The original MAC that was used when moving a physical device into an instance.

volatile.<name>.last_state.mtu Network device original MTU (page 444)

Key:	volatile.<name>.last_state.mtu
Type:	string

The original MTU that was used when moving a physical device into an instance.

volatile.<name>.last_state.vdpa.name VDPA device name (page 444)

Key:	volatile.<name>.last_state.vdpa.name
Type:	string

The VDPA device name used when moving a VDPA device file descriptor into an instance.

volatile.<name>.last_state.vf.hwaddr SR-IOV virtual function original MAC (page 444)

Key:	volatile.<name>.last_state.vf.hwaddr
Type:	string

The original MAC used when moving a VF into an instance.

volatile.<name>.last_state.vf.id SR-IOV virtual function ID (page 444)

Key:	volatile.<name>.last_state.vf.id
Type:	string

The ID used when moving a VF into an instance.

volatile.<name>.last_state.vf.spoofcheck SR-IOV virtual function original spoof check setting (page 444)

Key:	`volatile.<name>.last_state.vf.spoofcheck`
Type:	string

The original spoof check setting used when moving a VF into an instance.

`volatile.<name>.last_state.vf.vlan` SR-IOV virtual function original VLAN (page 445)

Key:	`volatile.<name>.last_state.vf.vlan`
Type:	string

The original VLAN used when moving a VF into an instance.

`volatile.apply_nvram` Whether to regenerate VM NVRAM the next time the instance starts (page 445)

Key:	`volatile.apply_nvram`
Type:	bool

`volatile.apply_template` Template hook (page 445)

Key:	`volatile.apply_template`
Type:	string

The template with the given name is triggered upon next startup.

`volatile.base_image` Hash of the base image (page 445)

Key:	`volatile.base_image`
Type:	string

The hash of the image that the instance was created from (empty if the instance was not created from an image).

`volatile.cloud-init.instance-id` `instance-id` (UUID) exposed to `cloud-init` (page 445)

Key:	`volatile.cloud-init.instance-id`
Type:	string

`volatile.evacuate.origin` The origin of the evacuated instance (page 445)

Key:	`volatile.evacuate.origin`
Type:	string

The cluster member that the instance lived on before evacuation.

`volatile.idmap.base` The first ID in the container's primary idmap range (page 446)

Key:	`volatile.idmap.base`
Type:	integer
Condition:	container

`volatile.idmap.current` The idmap currently in use by the container (page 446)

Key:	`volatile.idmap.current`
Type:	string
Condition:	container

`volatile.idmap.next` The idmap to use the next time the container starts (page 446)

Key:	`volatile.idmap.next`
Type:	string
Condition:	container

`volatile.last_state.idmap` On-disk UID/GID map for the container's rootfs (page 446)

Key:	`volatile.last_state.idmap`
Type:	string
Condition:	container

The UID/GID map that has been applied to the container's underlying storage. This is usually set for containers created on older kernels that don't support idmapped mounts.

`volatile.last_state.power` Instance state as of last host shutdown (page 446)

Key:	`volatile.last_state.power`
Type:	string

`volatile.uuid` Instance UUID (page 446)

Key:	`volatile.uuid`
Type:	string

The instance UUID is globally unique across all servers and projects.

`volatile.uuid.generation` Instance generation UUID (page 447)

Key:	`volatile.uuid.generation`
Type:	string

The instance generation UUID changes whenever the instance's place in time moves backwards. It is globally unique across all servers and projects.

`volatile.vsock_id` Instance `vsock ID` used as of last start (page 447)

Key:	`volatile.vsock_id`
Type:	string

Devices

Devices are attached to an instance (see *Configure devices* (page 87)) or to a profile (see *Edit a profile* (page 98)).

They include, for example, network interfaces, mount points, USB and GPU devices. These devices can have instance device options, depending on the type of the instance device.

LXD supports the following device types:

ID (database)	Name	Condition	Description
0	*none* (page 448)	-	Inheritance blocker
1	*nic* (page 449)	-	Network interface
2	*disk* (page 478)	-	Mount point inside the instance
3	*unix-char* (page 486)	container	Unix character device
4	*unix-block* (page 488)	container	Unix block device
5	*usb* (page 490)	-	USB device
6	*gpu* (page 491)	-	GPU device
7	*infiniband* (page 498)	container	InfiniBand device
8	*proxy* (page 499)	container	Proxy device
9	*unix-hotplug* (page 503)	container	Unix hotplug device
10	*tpm* (page 505)	-	TPM device
11	*pci* (page 506)	VM	PCI device

Each instance comes with a set of *Standard devices* (page 448).

Standard devices

LXD provides each instance with the basic devices that are required for a standard POSIX system to work. These devices aren't visible in the instance or profile configuration, and they may not be overridden.

The standard devices are:

Device	Type of device
`/dev/null`	Character device
`/dev/zero`	Character device
`/dev/full`	Character device
`/dev/console`	Character device
`/dev/tty`	Character device
`/dev/random`	Character device
`/dev/urandom`	Character device
`/dev/net/tun`	Character device
`/dev/fuse`	Character device
`lo`	Network interface

Any other devices must be defined in the instance configuration or in one of the profiles used by the instance. The default profile typically contains a network interface that becomes `eth0` in the instance.

Type: `none`

> **Note**
>
> The `none` device type is supported for both containers and VMs.

A `none` device doesn't have any properties and doesn't create anything inside the instance.

Its only purpose is to stop inheriting devices that come from profiles. To do so, add a device with the same name as the one that you do not want to inherit, but with the device type `none`.

You can add this device either in a profile that is applied after the profile that contains the original device, or directly on the instance.

Configuration examples

Add a `none` device to an instance:

```
lxc config device add <instance_name> <device_name> none
```

See *Configure devices* (page 87) for more information.

Type: `nic`

> **Note**
>
> The `nic` device type is supported for both containers and VMs.
>
> NICs support hotplugging for both containers and VMs (with the exception of the `ipvlan` NIC type).

Network devices, also referred to as *Network Interface Controllers* or *NICs*, supply a connection to a network. LXD supports several different types of network devices (*NIC types*).

`nictype` vs. `network`

When adding a network device to an instance, there are two methods to specify the type of device that you want to add: through the `nictype` device option or the `network` device option.

These two device options are mutually exclusive, and you can specify only one of them when you create a device. However, note that when you specify the `network` option, the `nictype` option is derived automatically from the network type.

`nictype`

When using the `nictype` device option, you can specify a network interface that is not controlled by LXD. Therefore, you must specify all information that LXD needs to use the network interface.

When using this method, the `nictype` option must be specified when creating the device, and it cannot be changed later.

`network`

When using the `network` device option, the NIC is linked to an existing *managed network* (page 354). In this case, LXD has all required information about the network, and you need to specify only the network name when adding the device.

When using this method, LXD derives the `nictype` option automatically. The value is read-only and cannot be changed.

Other device options that are inherited from the network are marked with a "yes" in the "Managed" field of the NIC-specific device options. You cannot customize these options directly for the NIC if you're using the `network` method.

See *Networking setups* (page 353) for more information.

Available NIC types

The following NICs can be added using the `nictype` or `network` options:

- *`bridged`* (page 450): Uses an existing bridge on the host and creates a virtual device pair to connect the host bridge to the instance.
- *`macvlan`* (page 455): Sets up a new network device based on an existing one, but using a different MAC address.
- *`sriov`* (page 457): Passes a virtual function of an SR-IOV-enabled physical network device into the instance.

- *physical* (page 460): Passes a physical device from the host through to the instance. The targeted device will vanish from the host and appear in the instance.

The following NICs can be added using only the `network` option:

- *ovn* (page 462): Uses an existing OVN network and creates a virtual device pair to connect the instance to it.

The following NICs can be added using only the `nictype` option:

- *ipvlan* (page 467): Sets up a new network device based on an existing one, using the same MAC address but a different IP.
- *p2p* (page 470): Creates a virtual device pair, putting one side in the instance and leaving the other side on the host.
- *routed* (page 472): Creates a virtual device pair to connect the host to the instance and sets up static routes and proxy ARP/NDP entries to allow the instance to join the network of a designated parent interface.

The available device options depend on the NIC type and are listed in the following sections.

nictype: bridged

> **Note**
>
> You can select this NIC type through the `nictype` option or the `network` option (see *Bridge network* (page 573) for information about the managed `bridge` network).

A `bridged` NIC uses an existing bridge on the host and creates a virtual device pair to connect the host bridge to the instance.

Device options

NIC devices of type `bridged` have the following device options: `boot.priority` Boot priority for VMs (page 450)

Key:	`boot.priority`
Type:	integer
Managed:	no

A higher value for this option means that the VM boots first.

`host_name` Name of the interface inside the host (page 450)

Key:	`host_name`
Type:	string
Default:	randomly assigned
Managed:	no

`hwaddr` MAC address of the new interface (page 450)

Key:	hwaddr
Type:	string
Default:	randomly assigned
Managed:	no

ipv4.address IPv4 address to assign to the instance through DHCP (page 451)

Key:	ipv4.address
Type:	string
Managed:	no

Set this option to none to restrict all IPv4 traffic when *security.ipv4_filtering* (page 454) is set.

ipv4.routes IPv4 static routes for the NIC to add on the host (page 451)

Key:	ipv4.routes
Type:	string
Managed:	no

Specify a comma-delimited list of IPv4 static routes for this NIC to add on the host.

ipv4.routes.external IPv4 static routes to route to NIC (page 451)

Key:	ipv4.routes.external
Type:	string
Managed:	no

Specify a comma-delimited list of IPv4 static routes to route to the NIC and publish on the uplink network (BGP).

ipv6.address IPv6 address to assign to the instance through DHCP (page 451)

Key:	ipv6.address
Type:	string
Managed:	no

Set this option to none to restrict all IPv6 traffic when *security.ipv6_filtering* (page 454) is set.

ipv6.routes IPv6 static routes for the NIC to add on the host (page 451)

Key:	ipv6.routes
Type:	string
Managed:	no

Specify a comma-delimited list of IPv6 static routes for this NIC to add on the host.

`ipv6.routes.external` IPv6 static routes to route to NIC (page 452)

Key:	`ipv6.routes.external`
Type:	string
Managed:	no

Specify a comma-delimited list of IPv6 static routes to route to the NIC and publish on the uplink network (BGP).

`limits.egress` I/O limit for outgoing traffic (page 452)

Key:	`limits.egress`
Type:	string
Managed:	no

Specify the limit in bit/s. Various suffixes are supported (see *Units for storage and network limits* (page 506)).

`limits.ingress` I/O limit for incoming traffic (page 452)

Key:	`limits.ingress`
Type:	string
Managed:	no

Specify the limit in bit/s. Various suffixes are supported (see *Units for storage and network limits* (page 506)).

`limits.max` I/O limit for both incoming and outgoing traffic (page 452)

Key:	`limits.max`
Type:	string
Managed:	no

This option is the same as setting both *`limits.ingress`* (page 452) and *`limits.egress`* (page 452).

Specify the limit in bit/s. Various suffixes are supported (see *Units for storage and network limits* (page 506)).

`limits.priority` `skb->priority` value for outgoing traffic (page 452)

Key:	`limits.priority`
Type:	integer
Managed:	no

The skb->priority value for outgoing traffic is used by the kernel queuing discipline (qdisc) to prioritize network packets. Specify the value as a 32-bit unsigned integer.

The effect of this value depends on the particular qdisc implementation, for example, SKBPRIO or QFQ. Consult the kernel qdisc documentation before setting this value.

maas.subnet.ipv4 MAAS IPv4 subnet to register the instance in (page 453)

Key:	maas.subnet.ipv4
Type:	string
Managed:	yes

maas.subnet.ipv6 MAAS IPv6 subnet to register the instance in (page 453)

Key:	maas.subnet.ipv6
Type:	string
Managed:	yes

mtu MTU of the new interface (page 453)

Key:	mtu
Type:	integer
Default:	parent MTU
Managed:	yes

name Name of the interface inside the instance (page 453)

Key:	name
Type:	string
Default:	kernel assigned
Managed:	no

network Managed network to link the device to (page 453)

Key:	network
Type:	string
Managed:	no

You can specify this option instead of specifying the nictype directly.

parent Name of the host device (page 453)

Key:	parent
Type:	string
Managed:	yes
Required:	if specifying the nictype directly

`queue.tx.length` Transmit queue length for the NIC (page 454)

Key:	`queue.tx.length`
Type:	integer
Managed:	no

`security.ipv4_filtering` Whether to prevent the instance from spoofing an IPv4 address (page 454)

Key:	`security.ipv4_filtering`
Type:	bool
Default:	`false`
Managed:	no

Set this option to `true` to prevent the instance from spoofing another instance's IPv4 address. This option enables *security.mac_filtering* (page 454).

`security.ipv6_filtering` Whether to prevent the instance from spoofing an IPv6 address (page 454)

Key:	`security.ipv6_filtering`
Type:	bool
Default:	`false`
Managed:	no

Set this option to `true` to prevent the instance from spoofing another instance's IPv6 address. This option enables *security.mac_filtering* (page 454).

`security.mac_filtering` Whether to prevent the instance from spoofing a MAC address (page 454)

Key:	`security.mac_filtering`
Type:	bool
Default:	`false`
Managed:	no

Set this option to `true` to prevent the instance from spoofing another instance's MAC address.

`security.port_isolation` Whether to respect port isolation (page 454)

Key:	`security.port_isolation`
Type:	bool
Default:	`false`
Managed:	no

Set this option to `true` to prevent the NIC from communicating with other NICs in the network that have port isolation enabled.

`vlan` VLAN ID to use for non-tagged traffic (page 455)

Key:	`vlan`
Type:	integer
Managed:	no

Set this option to `none` to remove the port from the default VLAN.

`vlan.tagged` VLAN IDs or VLAN ranges to join for tagged traffic (page 455)

Key:	`vlan.tagged`
Type:	integer
Managed:	no

Specify the VLAN IDs or ranges as a comma-delimited list.

Configuration examples

Add a `bridged` network device to an instance, connecting to a LXD managed network:

```
lxc network create <network_name> --type=bridge
lxc config device add <instance_name> <device_name> nic network=<network_name>
```

Note that `bridge` is the type when creating a managed bridge network, while the device `nictype` that is required when connecting to an unmanaged bridge is `bridged`.

Add a `bridged` network device to an instance, connecting to an existing bridge interface with `nictype`:

```
lxc config device add <instance_name> <device_name> nic nictype=bridged parent=
<existing_bridge>
```

See *How to create a network* (page 210) and *Configure devices* (page 87) for more information.

`nictype: macvlan`

> **Note**
>
> You can select this NIC type through the `nictype` option or the `network` option (see *Macvlan network* (page 593) for information about the managed `macvlan` network).

A `macvlan` NIC sets up a new network device based on an existing one, but using a different MAC address.

If you are using a `macvlan` NIC, communication between the LXD host and the instances is not possible. Both the host and the instances can talk to the gateway, but they cannot communicate directly.

Device options

NIC devices of type `macvlan` have the following device options: `boot.priority` Boot priority for VMs (page 456)

Key:	`boot.priority`
Type:	integer
Managed:	no

A higher value for this option means that the VM boots first.

`gvrp` Whether to use GARP VLAN Registration Protocol (page 456)

Key:	`gvrp`
Type:	bool
Default:	`false`
Managed:	no

This option specifies whether to register the VLAN using the GARP VLAN Registration Protocol.

`hwaddr` MAC address of the new interface (page 456)

Key:	`hwaddr`
Type:	string
Default:	randomly assigned
Managed:	no

`maas.subnet.ipv4` MAAS IPv4 subnet to register the instance in (page 456)

Key:	`maas.subnet.ipv4`
Type:	string
Managed:	yes

`maas.subnet.ipv6` MAAS IPv6 subnet to register the instance in (page 456)

Key:	`maas.subnet.ipv6`
Type:	string
Managed:	yes

`mtu` MTU of the new interface (page 456)

Key:	`mtu`
Type:	integer
Default:	parent MTU
Managed:	yes

`name` Name of the interface inside the instance (page 457)

Key:	`name`
Type:	string
Default:	kernel assigned
Managed:	no

`network` Managed network to link the device to (page 457)

Key:	`network`
Type:	string
Managed:	no

You can specify this option instead of specifying the `nictype` directly.

`parent` Name of the host device (page 457)

Key:	`parent`
Type:	string
Managed:	yes
Required:	if specifying the `nictype` directly

`vlan` VLAN ID to attach to (page 457)

Key:	`vlan`
Type:	integer
Managed:	no

Configuration examples

Add a `macvlan` network device to an instance, connecting to a LXD managed network:

```
lxc network create <network_name> --type=macvlan parent=<existing_NIC>
lxc config device add <instance_name> <device_name> nic network=<network_name>
```

Add a `macvlan` network device to an instance, connecting to an existing network interface with `nictype`:

```
lxc config device add <instance_name> <device_name> nic nictype=macvlan parent=
<existing_NIC>
```

See *How to create a network* (page 210) and *Configure devices* (page 87) for more information.

`nictype: sriov`

> **Note**
>
> You can select this NIC type through the `nictype` option or the `network` option (see *SR-IOV network* (page 600) for information about the managed `sriov` network).

An `sriov` NIC passes a virtual function of an SR-IOV-enabled physical network device into the instance.

An SR-IOV-enabled network device associates a set of virtual functions (VFs) with the single physical function (PF) of the network device. PFs are standard PCIe functions. VFs, on the other hand, are very lightweight PCIe functions that are optimized for data movement. They come with a limited set of configuration capabilities to prevent changing properties of the PF.

Given that VFs appear as regular PCIe devices to the system, they can be passed to instances just like a regular physical device.

VF allocation

The `sriov` interface type expects to be passed the name of an SR-IOV enabled network device on the system via the `parent` property. LXD then checks for any available VFs on the system.

By default, LXD allocates the first free VF it finds. If it detects that either none are enabled or all currently enabled VFs are in use, it bumps the number of supported VFs to the maximum value and uses the first free VF. If all possible VFs are in use or the kernel or card doesn't support incrementing the number of VFs, LXD returns an error.

> **Note**
>
> If you need LXD to use a specific VF, use a `physical` NIC instead of a `sriov` NIC and set its `parent` option to the VF name.

Device options

NIC devices of type `sriov` have the following device options: `boot.priority` Boot priority for VMs (page 458)

Key:	`boot.priority`
Type:	integer
Managed:	no

A higher value for this option means that the VM boots first.

`hwaddr` MAC address of the new interface (page 458)

Key:	`hwaddr`
Type:	string
Default:	randomly assigned
Managed:	no

`maas.subnet.ipv4` MAAS IPv4 subnet to register the instance in (page 458)

Key:	`maas.subnet.ipv4`
Type:	string
Managed:	yes

`maas.subnet.ipv6` MAAS IPv6 subnet to register the instance in (page 459)

Key:	`maas.subnet.ipv6`
Type:	string
Managed:	yes

`mtu` MTU of the new interface (page 459)

Key:	`mtu`
Type:	integer
Default:	kernel assigned
Managed:	yes

`name` Name of the interface inside the instance (page 459)

Key:	`name`
Type:	string
Default:	kernel assigned
Managed:	no

`network` Managed network to link the device to (page 459)

Key:	`network`
Type:	string
Managed:	no

You can specify this option instead of specifying the `nictype` directly.

`parent` Name of the host device (page 459)

Key:	`parent`
Type:	string
Managed:	yes
Required:	if specifying the `nictype` directly

`security.mac_filtering` Whether to prevent the instance from spoofing a MAC address (page 459)

Key:	security.mac_filtering
Type:	bool
Default:	false
Managed:	no

Set this option to true to prevent the instance from spoofing another instance's MAC address.

vlan VLAN ID to attach to (page 460)

Key:	vlan
Type:	integer
Managed:	no

Configuration examples

Add a sriov network device to an instance, connecting to a LXD managed network:

```
lxc network create <network_name> --type=sriov parent=<sriov_enabled_NIC>
lxc config device add <instance_name> <device_name> nic network=<network_name>
```

Add a sriov network device to an instance, connecting to an existing SR-IOV-enabled interface with nictype:

```
lxc config device add <instance_name> <device_name> nic nictype=sriov parent=
<sriov_enabled_NIC>
```

See *How to create a network* (page 210) and *Configure devices* (page 87) for more information.

nictype: physical

Note

- You can select this NIC type through the nictype option or the network option (see *Physical network* (page 595) for information about the managed physical network).
- You can have only one physical NIC for each parent device.

A physical NIC provides straight physical device pass-through from the host. The targeted device will vanish from the host and appear in the instance (which means that you can have only one physical NIC for each targeted device).

Device options

NIC devices of type physical have the following device options: boot.priority Boot priority for VMs (page 460)

Key:	`boot.priority`
Type:	integer
Managed:	no

A higher value for this option means that the VM boots first.

`gvrp` Whether to use GARP VLAN Registration Protocol (page 461)

Key:	`gvrp`
Type:	bool
Default:	`false`
Managed:	no

This option specifies whether to register the VLAN using the GARP VLAN Registration Protocol.

`hwaddr` MAC address of the new interface (page 461)

Key:	`hwaddr`
Type:	string
Default:	parent MAC address
Condition:	container
Managed:	no

`maas.subnet.ipv4` MAAS IPv4 subnet to register the instance in (page 461)

Key:	`maas.subnet.ipv4`
Type:	string
Managed:	no

`maas.subnet.ipv6` MAAS IPv6 subnet to register the instance in (page 461)

Key:	`maas.subnet.ipv6`
Type:	string
Managed:	no

`mtu` MTU of the new interface (page 461)

Key:	`mtu`
Type:	integer
Default:	parent MTU
Condition:	container
Managed:	no

`name` Name of the interface inside the instance (page 461)

Key:	`name`
Type:	string
Default:	kernel assigned
Managed:	no

`network` Managed network to link the device to (page 462)

Key:	`network`
Type:	string
Managed:	no

You can specify this option instead of specifying the `nictype` directly.

`parent` Name of the host device (page 462)

Key:	`parent`
Type:	string
Managed:	yes
Required:	if specifying the `nictype` directly

`vlan` VLAN ID to attach to (page 462)

Key:	`vlan`
Type:	integer
Condition:	container
Managed:	no

Configuration examples

Add a `physical` network device to an instance, connecting to an existing physical network interface with `nictype`:

```
lxc config device add <instance_name> <device_name> nic nictype=physical parent=
<physical_NIC>
```

Adding a `physical` network device to an instance using a managed network is not possible, because the `physical` managed network type is intended to be used only with OVN networks.

See *Configure devices* (page 87) for more information.

nictype: ovn

> **Note**
>
> You can select this NIC type only through the `network` option (see *OVN network* (page 587) for information about the managed `ovn` network).

An ovn NIC uses an existing OVN network and creates a virtual device pair to connect the instance to it.

SR-IOV hardware acceleration

To use acceleration=sriov, you must have a compatible SR-IOV physical NIC that supports the Ethernet switch device driver model (switchdev) in your LXD host. LXD assumes that the physical NIC (PF) is configured in switchdev mode and connected to the OVN integration OVS bridge, and that it has one or more virtual functions (VFs) active.

To achieve this, follow these basic prerequisite setup steps:

1. Set up PF and VF:

   1. Activate some VFs on PF (called enp9s0f0np0 in the following example, with a PCI address of 0000:09:00.0) and unbind them.
   2. Enable switchdev mode and hw-tc-offload on the PF.
   3. Rebind the VFs.

   ```
   echo 4 > /sys/bus/pci/devices/0000:09:00.0/sriov_numvfs
   for i in $(lspci -nnn | grep "Virtual Function" | cut -d' ' -f1); do echo
   0000:$i > /sys/bus/pci/drivers/mlx5_core/unbind; done
   devlink dev eswitch set pci/0000:09:00.0 mode switchdev
   ethtool -K enp9s0f0np0 hw-tc-offload on
   for i in $(lspci -nnn | grep "Virtual Function" | cut -d' ' -f1); do echo
   0000:$i > /sys/bus/pci/drivers/mlx5_core/bind; done
   ```

2. Set up OVS by enabling hardware offload and adding the PF NIC to the integration bridge (normally called br-int):

   ```
   ovs-vsctl set open_vswitch . other_config:hw-offload=true
   systemctl restart openvswitch-switch
   ovs-vsctl add-port br-int enp9s0f0np0
   ip link set enp9s0f0np0 up
   ```

VDPA hardware acceleration

To use acceleration=vdpa, you must have a compatible VDPA physical NIC. The setup is the same as for SR-IOV hardware acceleration, except that you must also enable the vhost_vdpa module and check that you have some available VDPA management devices :

```
modprobe vhost_vdpa && vdpa mgmtdev show
```

Device options

NIC devices of type ovn have the following device options: acceleration Enable hardware offloading (page 463)

Key:	acceleration
Type:	string
Default:	none
Managed:	no

Possible values are none, sriov, or vdpa. See *SR-IOV hardware acceleration* (page 463) for more information.

boot.priority Boot priority for VMs (page 464)

Key:	boot.priority
Type:	integer
Managed:	no

A higher value for this option means that the VM boots first.

host_name Name of the interface inside the host (page 464)

Key:	host_name
Type:	string
Default:	randomly assigned
Managed:	no

hwaddr MAC address of the new interface (page 464)

Key:	hwaddr
Type:	string
Default:	randomly assigned
Managed:	no

ipv4.address IPv4 address to assign to the instance through DHCP (page 464)

Key:	ipv4.address
Type:	string
Managed:	no

ipv4.routes IPv4 static routes to route for the NIC (page 464)

Key:	ipv4.routes
Type:	string
Managed:	no

Specify a comma-delimited list of IPv4 static routes to route for this NIC.

ipv4.routes.external IPv4 static routes to route to NIC (page 464)

Key:	ipv4.routes.external
Type:	string
Managed:	no

Specify a comma-delimited list of IPv4 static routes to route to the NIC and publish on the uplink network.

`ipv6.address` IPv6 address to assign to the instance through DHCP (page 465)

Key:	`ipv6.address`
Type:	string
Managed:	no

`ipv6.routes` IPv6 static routes to route to the NIC (page 465)

Key:	`ipv6.routes`
Type:	string
Managed:	no

Specify a comma-delimited list of IPv6 static routes to route to the NIC.

`ipv6.routes.external` IPv6 static routes to route to NIC (page 465)

Key:	`ipv6.routes.external`
Type:	string
Managed:	no

Specify a comma-delimited list of IPv6 static routes to route to the NIC and publish on the uplink network.

`name` Name of the interface inside the instance (page 465)

Key:	`name`
Type:	string
Default:	kernel assigned
Managed:	no

`nested` Parent NIC name to nest this NIC under (page 465)

Key:	`nested`
Type:	string
Managed:	no

See also *vlan* (page 466).

`network` Managed network to link the device to (page 465)

Key:	`network`
Type:	string
Managed:	yes
Required:	yes

`security.acls` Network ACLs to apply (page 466)

Key:	`security.acls`
Type:	string
Managed:	no

Specify a comma-separated list

`security.acls.default.egress.action` Default action to use for egress traffic (page 466)

Key:	`security.acls.default.egress.action`
Type:	string
Default:	`reject`
Managed:	no

The specified action is used for all egress traffic that doesn't match any ACL rule.

`security.acls.default.egress.logged` Whether to log egress traffic that doesn't match any ACL rule (page 466)

Key:	`security.acls.default.egress.logged`
Type:	bool
Default:	`false`
Managed:	no

`security.acls.default.ingress.action` Default action to use for ingress traffic (page 466)

Key:	`security.acls.default.ingress.action`
Type:	string
Default:	`reject`
Managed:	no

The specified action is used for all ingress traffic that doesn't match any ACL rule.

`security.acls.default.ingress.logged` Whether to log ingress traffic that doesn't match any ACL rule (page 466)

Key:	`security.acls.default.ingress.logged`
Type:	bool
Default:	`false`
Managed:	no

`vlan` VLAN ID to use when nesting (page 466)

Key:	vlan
Type:	integer
Managed:	no

See also *nested* (page 465).

Configuration examples

An `ovn` network device must be added using a managed network. To do so:

```
lxc network create <network_name> --type=ovn network=<parent_network>
lxc config device add <instance_name> <device_name> nic network=<network_name>
```

See *How to set up OVN with LXD* (page 266) for full instructions, and *How to create a network* (page 210) and *Configure devices* (page 87) for more information.

`nictype`: `ipvlan`

Note

- This NIC type is available only for containers, not for virtual machines.
- You can select this NIC type only through the `nictype` option.
- This NIC type does not support hotplugging.

An `ipvlan` NIC sets up a new network device based on an existing one, using the same MAC address but a different IP.

If you are using an `ipvlan` NIC, communication between the LXD host and the instances is not possible. Both the host and the instances can talk to the gateway, but they cannot communicate directly.

LXD currently supports IPVLAN in L2 and L3S mode. In this mode, the gateway is automatically set by LXD, but the IP addresses must be manually specified using the `ipv4.address` and/or `ipv6.address` options before the container is started.

DNS

The name servers must be configured inside the container, because they are not set automatically. To do this, set the following `sysctls`:

- When using IPv4 addresses:

  ```
  net.ipv4.conf.<parent>.forwarding=1
  ```

- When using IPv6 addresses:

  ```
  net.ipv6.conf.<parent>.forwarding=1
  net.ipv6.conf.<parent>.proxy_ndp=1
  ```

Device options

NIC devices of type `ipvlan` have the following device options: `gvrp` Whether to use GARP VLAN Registration Protocol (page 468)

Key:	`gvrp`
Type:	bool
Default:	`false`

This option specifies whether to register the VLAN using the GARP VLAN Registration Protocol.

`hwaddr` MAC address of the new interface (page 468)

Key:	`hwaddr`
Type:	string
Default:	randomly assigned

`ipv4.address` IPv4 static addresses to add to the instance (page 468)

Key:	`ipv4.address`
Type:	string

Specify a comma-delimited list of IPv4 static addresses to add to the instance. In `l2` mode, you can specify them as CIDR values or singular addresses using a subnet of `/24`.

`ipv4.gateway` IPv4 gateway (page 468)

Key:	`ipv4.gateway`
Type:	string
Default:	`auto` (`l3s`), `-` (`l2`)

In `l3s` mode, the option specifies whether to add an automatic default IPv4 gateway. Possible values are `auto` and `none`.

In `l2` mode, this option specifies the IPv4 address of the gateway.

`ipv4.host_table` Custom policy routing table ID to add IPv4 static routes to (page 468)

Key:	`ipv4.host_table`
Type:	integer

The custom policy routing table is in addition to the main routing table.

`ipv6.address` IPv6 static addresses to add to the instance (page 468)

Key:	`ipv6.address`
Type:	string

Specify a comma-delimited list of IPv6 static addresses to add to the instance. In `l2` mode, you can specify them as CIDR values or singular addresses using a subnet of `/64`.

`ipv6.gateway` IPv6 gateway (page 469)

Key:	`ipv6.gateway`
Type:	string
Default:	`auto` (`l3s`), - (`l2`)

In `l3s` mode, the option specifies whether to add an automatic default IPv6 gateway. Possible values are `auto` and `none`.

In `l2` mode, this option specifies the IPv6 address of the gateway.

`ipv6.host_table` Custom policy routing table ID to add IPv6 static routes to (page 469)

Key:	`ipv6.host_table`
Type:	integer

The custom policy routing table is in addition to the main routing table.

`mode` IPVLAN mode (page 469)

Key:	`mode`
Type:	string
Default:	`l3s`

Possible values are `l2` and `l3s`.

`mtu` The MTU of the new interface (page 469)

Key:	`mtu`
Type:	integer
Default:	parent MTU

`name` Name of the interface inside the instance (page 469)

Key:	`name`
Type:	string
Default:	kernel assigned

`parent` Name of the host device (page 469)

Key:	`parent`
Type:	string
Required:	yes

`vlan` VLAN ID to attach to (page 470)

Key:	`vlan`
Type:	integer

Configuration examples

Add an `ipvlan` network device to an instance, connecting to an existing network interface with `nictype`:

```
lxc stop <instance_name>
lxc config device add <instance_name> <device_name> nic nictype=ipvlan parent=
<existing_NIC>
```

Adding an `ipvlan` network device to an instance using a managed network is not possible.

See *Configure devices* (page 87) for more information.

`nictype`: `p2p`

Note

You can select this NIC type only through the `nictype` option.

A `p2p` NIC creates a virtual device pair, putting one side in the instance and leaving the other side on the host.

Device options

NIC devices of type `p2p` have the following device options: `boot.priority` Boot priority for VMs (page 470)

Key:	`boot.priority`
Type:	integer

A higher value for this option means that the VM boots first.

`host_name` Name of the interface inside the host (page 470)

Key:	`host_name`
Type:	string
Default:	randomly assigned

`hwaddr` MAC address of the new interface (page 470)

Key:	`hwaddr`
Type:	string
Default:	randomly assigned

`ipv4.routes` IPv4 static routes for the NIC to add on the host (page 471)

Key:	`ipv4.routes`
Type:	string

Specify a comma-delimited list of IPv4 static routes for this NIC to add on the host.

`ipv6.routes` IPv6 static routes for the NIC to add on the host (page 471)

Key:	`ipv6.routes`
Type:	string

Specify a comma-delimited list of IPv6 static routes for this NIC to add on the host.

`limits.egress` I/O limit for outgoing traffic (page 471)

Key:	`limits.egress`
Type:	string

Specify the limit in bit/s. Various suffixes are supported (see *Units for storage and network limits* (page 506)).

`limits.ingress` I/O limit for incoming traffic (page 471)

Key:	`limits.ingress`
Type:	string

Specify the limit in bit/s. Various suffixes are supported (see *Units for storage and network limits* (page 506)).

`limits.max` I/O limit for both incoming and outgoing traffic (page 471)

Key:	`limits.max`
Type:	string

This option is the same as setting both *`limits.ingress`* (page 452) and *`limits.egress`* (page 452).

Specify the limit in bit/s. Various suffixes are supported (see *Units for storage and network limits* (page 506)).

limits.priority skb->priority value for outgoing traffic (page 471)

Key:	limits.priority
Type:	integer

The skb->priority value for outgoing traffic is used by the kernel queuing discipline (qdisc) to prioritize network packets. Specify the value as a 32-bit unsigned integer.

The effect of this value depends on the particular qdisc implementation, for example, SKBPRIO or QFQ. Consult the kernel qdisc documentation before setting this value.

mtu MTU of the new interface (page 472)

Key:	mtu
Type:	integer
Default:	kernel assigned

name Name of the interface inside the instance (page 472)

Key:	name
Type:	string
Default:	kernel assigned

queue.tx.length Transmit queue length for the NIC (page 472)

Key:	queue.tx.length
Type:	integer

Configuration examples

Add a p2p network device to an instance using nictype:

```
lxc config device add <instance_name> <device_name> nic nictype=p2p
```

Adding a p2p network device to an instance using a managed network is not possible.

See *Configure devices* (page 87) for more information.

nictype: routed

> **Note**
>
> You can select this NIC type only through the nictype option.

A routed NIC creates a virtual device pair to connect the host to the instance and sets up static routes and proxy ARP/NDP entries to allow the instance to join the network of a designated parent interface. For containers it uses a virtual Ethernet device pair, and for VMs it uses a TAP device.

This NIC type is similar in operation to `ipvlan`, in that it allows an instance to join an external network without needing to configure a bridge and shares the host's MAC address. However, it differs from `ipvlan` because it does not need IPVLAN support in the kernel, and the host and the instance can communicate with each other.

This NIC type respects `netfilter` rules on the host and uses the host's routing table to route packets, which can be useful if the host is connected to multiple networks.

IP addresses, gateways and routes

You must manually specify the IP addresses (using `ipv4.address` and/or `ipv6.address`) before the instance is started.

For containers, the NIC configures the following link-local gateway IPs on the host end and sets them as the default gateways in the container's NIC interface:

```
169.254.0.1
fe80::1
```

For VMs, the gateways must be configured manually or via a mechanism like `cloud-init` (see the *how to guide* (page 123)).

> **Note**
>
> If your container image is configured to perform DHCP on the interface, it will likely remove the automatically added configuration. In this case, you must configure the IP addresses and gateways manually or via a mechanism like `cloud-init`.

The NIC type configures static routes on the host pointing to the instance's `veth` interface for all of the instance's IPs.

Multiple IP addresses

Each NIC device can have multiple IP addresses added to it.

However, it might be preferable to use multiple `routed` NIC interfaces instead. In this case, set the `ipv4.gateway` and `ipv6.gateway` values to `none` on any subsequent interfaces to avoid default gateway conflicts. Also consider specifying a different host-side address for these subsequent interfaces using `ipv4.host_address` and/or `ipv6.host_address`.

Parent interface

This NIC can operate with and without a `parent` network interface set.

With the `parent` network interface set, proxy ARP/NDP entries of the instance's IPs are added to the parent interface, which allows the instance to join the parent interface's network at layer 2.

To enable this, the following network configuration must be applied on the host via `sysctl`:

- When using IPv4 addresses:

```
net.ipv4.conf.<parent>.forwarding=1
```

- When using IPv6 addresses:

```
net.ipv6.conf.all.forwarding=1
net.ipv6.conf.<parent>.forwarding=1
net.ipv6.conf.all.proxy_ndp=1
net.ipv6.conf.<parent>.proxy_ndp=1
```

Device options

NIC devices of type `routed` have the following device options: `gvrp` Whether to use GARP VLAN Registration Protocol (page 474)

Key:	`gvrp`
Type:	bool
Default:	`false`

This option specifies whether to register the VLAN using the GARP VLAN Registration Protocol.

`host_name` Name of the interface inside the host (page 474)

Key:	`host_name`
Type:	string
Default:	randomly assigned

`hwaddr` MAC address of the new interface (page 474)

Key:	`hwaddr`
Type:	string
Default:	randomly assigned

`ipv4.address` IPv4 static addresses to add to the instance (page 474)

Key:	`ipv4.address`
Type:	string

Specify a comma-delimited list of IPv4 static addresses to add to the instance.

`ipv4.gateway` Whether to add an automatic default IPv4 gateway (page 474)

Key:	`ipv4.gateway`
Type:	string
Default:	`auto`

Possible values are `auto` and `none`.

ipv4.host_address IPv4 address to add to the host-side veth interface (page 474)

Key:	ipv4.host_address
Type:	string
Default:	169.254.0.1

ipv4.host_table Custom policy routing table ID to add IPv4 static routes to (page 475)

Key:	ipv4.host_table
Type:	integer

The custom policy routing table is in addition to the main routing table.

ipv4.neighbor_probe Whether to probe the parent network for IPv4 address availability (page 475)

Key:	ipv4.neighbor_probe
Type:	bool
Default:	true

ipv4.routes IPv4 static routes for the NIC to add on the host (page 475)

Key:	ipv4.routes
Type:	string

Specify a comma-delimited list of IPv4 static routes for this NIC to add on the host (without L2 ARP/NDP proxy).

ipv6.address IPv6 static addresses to add to the instance (page 475)

Key:	ipv6.address
Type:	string

Specify a comma-delimited list of IPv6 static addresses to add to the instance.

ipv6.gateway Whether to add an automatic default IPv6 gateway (page 475)

Key:	ipv6.gateway
Type:	string
Default:	auto

Possible values are auto and none.

ipv6.host_address IPv6 address to add to the host-side veth interface (page 475)

Key:	`ipv6.host_address`
Type:	string
Default:	`fe80::1`

`ipv6.host_table` Custom policy routing table ID to add IPv6 static routes to (page 476)

Key:	`ipv6.host_table`
Type:	integer

The custom policy routing table is in addition to the main routing table.

`ipv6.neighbor_probe` Whether to probe the parent network for IPv6 address availability (page 476)

Key:	`ipv6.neighbor_probe`
Type:	bool
Default:	`true`

`ipv6.routes` IPv6 static routes for the NIC to add on the host (page 476)

Key:	`ipv6.routes`
Type:	string

Specify a comma-delimited list of IPv6 static routes for this NIC to add on the host (without L2 ARP/NDP proxy).

`limits.egress` I/O limit for outgoing traffic (page 476)

Key:	`limits.egress`
Type:	string

Specify the limit in bit/s. Various suffixes are supported (see *Units for storage and network limits* (page 506)).

`limits.ingress` I/O limit for incoming traffic (page 476)

Key:	`limits.ingress`
Type:	string

Specify the limit in bit/s. Various suffixes are supported (see *Units for storage and network limits* (page 506)).

`limits.max` I/O limit for both incoming and outgoing traffic (page 476)

Key:	`limits.max`
Type:	string

This option is the same as setting both *limits.ingress* (page 452) and *limits.egress* (page 452).

Specify the limit in bit/s. Various suffixes are supported (see *Units for storage and network limits* (page 506)).

`limits.priority` `skb->priority` value for outgoing traffic (page 477)

Key:	`limits.priority`
Type:	integer

The `skb->priority` value for outgoing traffic is used by the kernel queuing discipline (qdisc) to prioritize network packets. Specify the value as a 32-bit unsigned integer.

The effect of this value depends on the particular qdisc implementation, for example, `SKBPRIO` or `QFQ`. Consult the kernel qdisc documentation before setting this value.

`mtu` The MTU of the new interface (page 477)

Key:	`mtu`
Type:	integer
Default:	parent MTU

`name` Name of the interface inside the instance (page 477)

Key:	`name`
Type:	string
Default:	kernel assigned

`parent` Name of the host device to join the instance to (page 477)

Key:	`parent`
Type:	string

`queue.tx.length` Transmit queue length for the NIC (page 477)

Key:	`queue.tx.length`
Type:	integer

`vlan` VLAN ID to attach to (page 477)

Key:	`vlan`
Type:	integer

Configuration examples

Add a `routed` network device to an instance using `nictype`:

```
lxc config device add <instance_name> <device_name> nic nictype=routed ipv4.
address=192.0.2.2 ipv6.address=2001:db8::2
```

Adding a `routed` network device to an instance using a managed network is not possible.

See *Configure devices* (page 87) for more information.

`bridged`, `macvlan` or `ipvlan` for connection to physical network

The `bridged`, `macvlan` and `ipvlan` interface types can be used to connect to an existing physical network.

`macvlan` effectively lets you fork your physical NIC, getting a second interface that is then used by the instance. This method saves you from creating a bridge device and virtual Ethernet device pairs and usually offers better performance than a bridge.

The downside to this method is that `macvlan` devices, while able to communicate between themselves and to the outside, cannot talk to their parent device. This means that you can't use `macvlan` if you ever need your instances to talk to the host itself.

In such case, a `bridge` device is preferable. A bridge also lets you use MAC filtering and I/O limits, which cannot be applied to a `macvlan` device.

`ipvlan` is similar to `macvlan`, with the difference being that the forked device has IPs statically assigned to it and inherits the parent's MAC address on the network.

MAAS integration

If you're using MAAS to manage the physical network under your LXD host and want to attach your instances directly to a MAAS-managed network, LXD can be configured to interact with MAAS so that it can track your instances.

At the daemon level, you must configure *`maas.api.url`* (page 412) and *`maas.api.key`* (page 412), and then set the NIC-specific `maas.subnet.ipv4` and/or `maas.subnet.ipv6` keys on the instance or profile's `nic` entry.

With this configuration, LXD registers all your instances with MAAS, giving them proper DHCP leases and DNS records.

If you set the `ipv4.address` or `ipv6.address` keys on the NIC, those are registered as static assignments in MAAS.

Type: `disk`

> **Note**
>
> The `disk` device type is supported for both containers and VMs. It supports hotplugging for both containers and VMs.

Disk devices supply additional storage to instances.

For containers, they are essentially mount points inside the instance (either as a bind-mount of an existing file or directory on the host, or, if the source is a block device, a regular mount). Virtual machines share host-side mounts or directories through `9p` or `virtiofs` (if available), or as VirtIO disks for block-based disks.

Types of disk devices

You can create disk devices from different sources. The value that you specify for the `source` option specifies the type of disk device that is added. See *Configuration examples* (page 485) for more detailed information on how to add each type of disk device.

Storage volume

The most common type of disk device is a storage volume. Specify the storage volume name as the *`source`* (page 484) to add a storage volume as a disk device. `virtual-machine' storage volumes (and their snapshots) can also be attached as disk devices.

Path on the host

You can share a path on your host (either a file system or a block device) to your instance. Specify the host path as the source to add it as a disk device.

Ceph RBD

LXD can use Ceph to manage an internal file system for the instance, but if you have an existing, externally managed Ceph RBD that you would like to use for an instance, you can add it by specifying `ceph:<pool_name>/<volume_name>` as the source.

CephFS

LXD can use Ceph to manage an internal file system for the instance, but if you have an existing, externally managed Ceph file system that you would like to use for an instance, you can add it by specifying `cephfs:<fs_name>/<path>` as the source.

ISO file

You can add an ISO file as a disk device for a virtual machine by specifying its file path as the source. It is added as a ROM device inside the VM.

This source type is applicable only to VMs.

VM `cloud-init`

You can generate a `cloud-init` configuration ISO from the *`cloud-init.vendor-data`* (page 420) and *`cloud-init.user-data`* (page 420) configuration keys and attach it to a virtual machine by specifying `cloud-init:config` as the source. The `cloud-init` that is running inside the VM then detects the drive on boot and applies the configuration.

This source type is applicable only to VMs.

Adding such a configuration disk might be needed if the VM image that is used includes `cloud-init` but not the `lxd-agent`. This is the case for official Ubuntu images prior to `20.04`. On such images, the following steps enable the LXD agent and thus provide the ability to use `lxc exec` to access the VM:

```
lxc init ubuntu-daily:18.04 --vm u1
lxc config device add u1 config disk source=cloud-init:config
lxc config set u1 cloud-init.user-data - << EOF
#cloud-config
#packages:
#  - linux-image-virtual-hwe-16.04  # 16.04 GA kernel as a problem with vsock
runcmd:
  - mount -t 9p config /mnt
  - cd /mnt
  - ./install.sh
  - cd /
  - umount /mnt
  - systemctl start lxd-agent  # XXX: causes a reboot
EOF
lxc start --console u1
```

Note that for `16.04`, the HWE kernel is required to work around a problem with `vsock` (see the commented out section in the above `cloud-config`).

Initial volume configuration for instance root disk devices

Initial volume configuration allows setting specific configurations for the root disk devices of new instances. These settings are prefixed with `initial.` and are only applied when the instance is created. This method allows creating instances that have unique configurations, independent of the default storage pool settings.

For example, you can add an initial volume configuration for *zfs.block_mode* (page 569) to an existing profile, and this will then take effect for each new instance you create using this profile:

```
lxc profile device set <profile_name> <device_name> initial.zfs.block_mode=true
```

You can also set an initial configuration directly when creating an instance. For example:

```
lxc init <image> <instance_name> --device <device_name>,initial.zfs.block_
mode=true
```

Note that you cannot use initial volume configurations with custom volume options or to set the volume's size (quota).

Device options

`disk` devices have the following device options: `boot.priority` Boot priority for VMs (page 480)

Key:	`boot.priority`
Type:	integer
Condition:	virtual machine
Required:	no

A higher value indicates a higher boot precedence for the disk device. This is useful for prioritizing boot sources like ISO-backed disks.

ceph.cluster_name Cluster name of the Ceph cluster (page 481)

Key:	ceph.cluster_name
Type:	string
Default:	ceph
Required:	for Ceph or CephFS sources

ceph.user_name User name of the Ceph cluster (page 481)

Key:	ceph.user_name
Type:	string
Default:	admin
Required:	for Ceph or CephFS sources

initial.* Initial volume configuration (page 481)

Key:	initial.*
Type:	n/a
Required:	no

Initial volume configuration allows setting unique configurations independent of the default storage pool settings. See *Initial volume configuration for instance root disk devices* (page 480) for more information.

io.bus Bus for the device (page 481)

Key:	io.bus
Type:	string
Default:	virtio-scsi
Condition:	virtual machine
Required:	no

Possible values are virtio-scsi, virtio-blk or nvme.

io.cache Caching mode for the device (page 481)

Key:	io.cache
Type:	string
Default:	none
Condition:	virtual machine
Required:	no

Possible values are none, writeback, or unsafe.

io.threads Thread pool for virtiofs file system shares (page 481)

Key:	`io.threads`
Type:	integer
Default:	`0`
Condition:	virtual machine
Required:	no

This option controls the `virtiofsd` thread pool size, which can help improve I/O performance. Only applies to virtiofs file system shares. In *`restricted`* (page 514) projects, it can only be used when *`restricted.virtual-machines.lowlevel`* (page 520) is set to `allow`.

`limits.max` I/O limit in byte/s or IOPS for both read and write (page 482)

Key:	`limits.max`
Type:	string
Required:	no

This option is the same as setting both *`limits.read`* (page 482) and *`limits.write`* (page 482).

You can specify a value in byte/s (various suffixes supported, see *Units for storage and network limits* (page 506)) or in IOPS (must be suffixed with `iops`). See also *Configure I/O options* (page 188).

`limits.read` Read I/O limit in byte/s or IOPS (page 482)

Key:	`limits.read`
Type:	string
Required:	no

You can specify a value in byte/s (various suffixes supported, see *Units for storage and network limits* (page 506)) or in IOPS (must be suffixed with `iops`). See also *Configure I/O options* (page 188).

`limits.write` Write I/O limit in byte/s or IOPS (page 482)

Key:	`limits.write`
Type:	string
Required:	no

You can specify a value in byte/s (various suffixes supported, see *Units for storage and network limits* (page 506)) or in IOPS (must be suffixed with `iops`). See also *Configure I/O options* (page 188).

`path` Mount path (page 482)

Key:	`path`
Type:	string
Condition:	container
Required:	yes

This option specifies the path inside the container where the disk will be mounted.

`pool` Storage pool to which the disk device belongs (page 483)

Key:	`pool`
Type:	string
Condition:	storage volumes managed by LXD
Required:	no

`propagation` How a bind-mount is shared between the instance and the host (page 483)

Key:	`propagation`
Type:	string
Default:	`private`
Required:	no

Possible values are `private` (the default), `shared`, `slave`, `unbindable`, `rshared`, `rslave`, `runbindable`, `rprivate`. See the Linux Kernel shared subtree[245] documentation for a full explanation.

`raw.mount.options` File system specific mount options (page 483)

Key:	`raw.mount.options`
Type:	string
Required:	no

`readonly` Whether to make the mount read-only (page 483)

Key:	`readonly`
Type:	bool
Default:	`false`
Required:	no

`recursive` Whether to recursively mount the source path (page 483)

Key:	`recursive`
Type:	bool
Default:	`false`
Required:	no

[245] https://www.kernel.org/doc/Documentation/filesystems/sharedsubtree.txt

`required` Whether to fail if the source doesn't exist (page 483)

Key:	`required`
Type:	bool
Default:	`true`
Required:	no

`shift` Whether to set up a UID/GID shifting overlay (page 484)

Key:	`shift`
Type:	bool
Default:	`false`
Condition:	container
Required:	no

If enabled, this option sets up a shifting overlay to translate the source UID/GID to match the container instance.

`size` Disk size (page 484)

Key:	`size`
Type:	string
Required:	no

This option is supported only for the rootfs (/).

Specify a value in bytes (various suffixes supported, see *Units for storage and network limits* (page 506)).

`size.state` Size of the file-system volume used for saving runtime state (page 484)

Key:	`size.state`
Type:	string
Condition:	virtual machine
Required:	no

This option is similar to *`size`* (page 484), but applies to the file-system volume used for saving the runtime state in VMs.

`source` Source of a file system or block device (page 484)

Key:	`source`
Type:	string
Required:	yes

See *Types of disk devices* (page 479) for details.

`source.snapshot` `source` snapshot name (page 484)

Key:	`source.snapshot`
Type:	string
Required:	no

Snapshot of the volume given by `source`.

`source.type` Type of the backing storage volume (page 485)

Key:	`source.type`
Type:	string
Default:	`custom`
Required:	no

Possible values are `custom` (the default) or `virtual-machine`. This key is only valid when `source` is the name of a storage volume.

Configuration examples

How to add a disk device depends on its *type* (page 479).

Storage volume

To add a storage volume, specify its name as the `source` of the device:

```
lxc config device add <instance_name> <device_name> disk pool=<pool_name>
source=<volume_name> [path=<path_in_instance>]
```

The path is required for file system volumes, but not for block volumes.

Alternatively, you can use the *`lxc storage volume attach`* (page 898) command to *Attach the volume to an instance* (page 186). Both commands use the same mechanism to add a storage volume as a disk device.

Path on the host

To add a host device, specify the host path as the `source`:

```
lxc config device add <instance_name> <device_name> disk source=<path_on_
host> [path=<path_in_instance>]
```

The path is required for file systems, but not for block devices.

Ceph RBD

To add an existing Ceph RBD volume, specify its pool and volume name:

```
lxc config device add <instance_name> <device_name> disk source=ceph:<pool_
name>/<volume_name> ceph.user_name=<user_name> ceph.cluster_name=<cluster_
name> [path=<path_in_instance>]
```

The path is required for file systems, but not for block devices.

CephFS

To add an existing CephFS file system, specify its name and path:

```
lxc config device add <instance_name> <device_name> disk source=cephfs:<fs_name>/<path> ceph.user_name=<user_name> ceph.cluster_name=<cluster_name> path=<path_in_instance>
```

ISO file

To add an ISO file, specify its file path as the `source`:

```
lxc config device add <instance_name> <device_name> disk source=<file_path_on_host>
```

VM `cloud-init`

To add `cloud-init` configuration, specify `cloud-init:config` as the source:

```
lxc config device add <instance_name> <device_name> disk source=cloud-init:config
```

See *Configure devices* (page 87) for more information.

Type: `unix-char`

> **Note**
>
> The `unix-char` device type is supported for containers. It supports hotplugging.

Unix character devices make the specified character device appear as a device in the container (under `/dev`). You can read from the device and write to it.

Device options

`unix-char` devices have the following device options: `gid` GID of the device owner in the container (page 486)

Key:	`gid`
Type:	integer
Default:	`0`

`major` Device major number (page 486)

Key:	`major`
Type:	integer
Default:	device on host

`minor` Device minor number (page 486)

Key:	`minor`
Type:	integer
Default:	device on host

`mode` Mode of the device in the container (page 486)

Key:	`mode`
Type:	integer
Default:	`0660`

`path` Path inside the container (page 487)

Key:	`path`
Type:	string
Required:	either `source` or `path` must be set

`required` Whether this device is required to start the container (page 487)

Key:	`required`
Type:	bool
Default:	`true`

See *Hotplugging* (page 488) for more information.

`source` Path on the host (page 487)

Key:	`source`
Type:	string
Required:	either `source` or `path` must be set

`uid` UID of the device owner in the container (page 487)

Key:	`uid`
Type:	integer
Default:	`0`

Configuration examples

Add a `unix-char` device to a container by specifying its source and path:

```
lxc config device add <instance_name> <device_name> unix-char source=<path_on_
host> path=<path_on_instance>
```

If you want to use the same path on the container as on the host, you can omit the `source` option:

```
lxc config device add <instance_name> <device_name> unix-char path=<path_to_the_
device>
```

See *Configure devices* (page 87) for more information.

Hotplugging

Hotplugging is enabled if you set `required=false` and specify the `source` option for the device.

In this case, the device is automatically passed into the container when it appears on the host, even after the container starts. If the device disappears from the host system, it is removed from the container as well.

Type: `unix-block`

> **Note**
>
> The `unix-block` device type is supported for containers. It supports hotplugging.

Unix block devices make the specified block device appear as a device in the container (under `/dev`). You can read from the device and write to it.

Device options

`unix-block` devices have the following device options: `gid` GID of the device owner in the container (page 488)

Key:	`gid`
Type:	integer
Default:	`0`

`major` Device major number (page 488)

Key:	`major`
Type:	integer
Default:	device on host

`minor` Device minor number (page 488)

Key:	`minor`
Type:	integer
Default:	device on host

`mode` Mode of the device in the container (page 488)

Key:	`mode`
Type:	integer
Default:	`0660`

`path` Path inside the container (page 488)

Key:	path
Type:	string
Required:	either source or path must be set

required Whether this device is required to start the container (page 489)

Key:	required
Type:	bool
Default:	true

See *Hotplugging* (page 489) for more information.

source Path on the host (page 489)

Key:	source
Type:	string
Required:	either source or path must be set

uid UID of the device owner in the container (page 489)

Key:	uid
Type:	integer
Default:	0

Configuration examples

Add a unix-block device to a container by specifying its source and path:

```
lxc config device add <instance_name> <device_name> unix-block source=<path_on_
host> path=<path_on_instance>
```

If you want to use the same path on the container as on the host, you can omit the source option:

```
lxc config device add <instance_name> <device_name> unix-block path=<path_to_the_
device>
```

See *Configure devices* (page 87) for more information.

Hotplugging

Hotplugging is enabled if you set required=false and specify the source option for the device.

In this case, the device is automatically passed into the container when it appears on the host, even after the container starts. If the device disappears from the host system, it is removed from the container as well.

Type: usb

> **Note**
>
> The usb device type is supported for both containers and VMs. It supports hotplugging for both containers and VMs.

USB devices make the specified USB device appear in the instance. For performance issues, avoid using devices that require high throughput or low latency.

For containers, only libusb devices (at /dev/bus/usb) are passed to the instance. This method works for devices that have user-space drivers. For devices that require dedicated kernel drivers, use a *unix-char device* (page 486) or a *unix-hotplug device* (page 503) instead.

For virtual machines, the entire USB device is passed through, so any USB device is supported. When a device is passed to the instance, it vanishes from the host.

Device options

usb devices have the following device options: busnum The bus number of which the USB device is attached (page 490)

Key:	busnum
Type:	int

devnum The device number of the USB device (page 490)

Key:	devnum
Type:	int

gid GID of the device owner in the instance (page 490)

Key:	gid
Type:	integer
Default:	0
Condition:	container

mode Mode of the device in the instance (page 490)

Key:	mode
Type:	integer
Default:	0660
Condition:	container

productid Product ID of the USB device (page 490)

Key:	productid
Type:	string

`required` Whether this device is required to start the instance (page 491)

Key:	`required`
Type:	bool
Default:	`false`

The default is `false`, which means that all devices can be hotplugged.

`serial` The serial number of the USB device (page 491)

Key:	`serial`
Type:	string

`uid` UID of the device owner in the instance (page 491)

Key:	`uid`
Type:	integer
Default:	`0`
Condition:	container

`vendorid` Vendor ID of the USB device (page 491)

Key:	`vendorid`
Type:	string

Configuration examples

Add a `usb` device to an instance by specifying its vendor ID and product ID:

```
lxc config device add <instance_name> <device_name> usb vendorid=<vendor_ID>
productid=<product_ID>
```

To determine the vendor ID and product ID, you can use **`lsusb`**, for example.

See *Configure devices* (page 87) for more information.

Type: `gpu`

GPU devices make the specified GPU device or devices appear in the instance.

> **Note**
>
> For containers, a `gpu` device may match multiple GPUs at once. For VMs, each device can match only a single GPU.

The following types of GPUs can be added using the `gputype` device option:

- *`physical`* (page 492) (container and VM): Passes an entire GPU through into the instance. This value is the default if `gputype` is unspecified.

- *mdev* (page 494) (VM only): Creates and passes a virtual GPU (vGPU) through into the instance.
- *mig* (page 495) (container only): Creates and passes a MIG (Multi-Instance GPU) through into the instance.
- *sriov* (page 497) (VM only): Passes a virtual function of an SR-IOV-enabled GPU into the instance.

The available device options depend on the GPU type and are listed in the tables in the following sections.

gputype: physical

> **Note**
>
> The `physical` GPU type is supported for both containers and VMs. It supports hotplugging only for containers, not for VMs.

A `physical` GPU device passes an entire GPU through into the instance.

Device options

GPU devices of type `physical` have the following device options: `gid` GID of the device owner in the container (page 492)

Key:	`gid`
Type:	integer
Default:	`0`
Condition:	container

`id` ID of the GPU device (page 492)

Key:	`id`
Type:	string

The ID can either be the DRM card ID of the GPU device (container or VM) or a fully-qualified Container Device Interface (CDI) name (container only). Here are some examples of fully-qualified CDI names:

- `nvidia.com/gpu=0`: Instructs LXD to operate a discrete GPU (dGPU) pass-through of brand NVIDIA with the first discovered GPU on your system. You can use the `nvidia-smi` tool on your host to find out which identifier to use.
- `nvidia.com/gpu=1833c8b5-9aa0-5382-b784-68b7e77eb185`: Instructs LXD to operate a discrete GPU (dGPU) pass-through of brand NVIDIA with a given GPU unique identifier. This identifier should also appear with `nvidia-smi -L`.
- `nvidia.com/igpu=all`: Instructs LXD to pass all the host integrated GPUs (iGPU) of brand NVIDIA. The concept of an index does not currently map to iGPUs. It is possible

to list them with the `nvidia-smi -L` command. A special `nvgpu` mention should appear in the generated list to indicate a device to be an iGPU.

- `nvidia.com/gpu=all`: Instructs LXD to pass all the host GPUs of brand NVIDIA through to the container.

`mode` Mode of the device in the container (page 493)

Key:	`mode`
Type:	integer
Default:	`0660`
Condition:	container

`pci` PCI address of the GPU device (page 493)

Key:	`pci`
Type:	string

`productid` Product ID of the GPU device (page 493)

Key:	`productid`
Type:	string

`uid` UID of the device owner in the container (page 493)

Key:	`uid`
Type:	integer
Default:	`0`
Condition:	container

`vendorid` Vendor ID of the GPU device (page 493)

Key:	`vendorid`
Type:	string

Configuration examples

Add all GPUs from the host system as a `physical` GPU device to an instance:

```
lxc config device add <instance_name> <device_name> gpu gputype=physical
```

Add a specific GPU from the host system as a `physical` GPU device to an instance by specifying its PCI address:

```
lxc config device add <instance_name> <device_name> gpu gputype=physical pci=<pci_
address>
```

See *Configure devices* (page 87) for more information.

CDI mode

> **Note**
>
> The CDI mode is currently not supported on `armhf` architectures.

Add a specific GPU from the host system as a `physical` GPU device to an instance using the Container Device Interface[246] (CDI) notation through a fully-qualified CDI name:

```
lxc config device add &lt;instance_name&gt; &lt;device_name&gt; gpu gputype=physical id=
<fully_qualified_CDI_name>
```

For example, add the first available NVIDIA discrete GPU on your system:

```
lxc config device add <instance_name> <device_name> gpu gputype=physical
id=nvidia.com/gpu=0
```

If your machine has an NVIDIA iGPU (integrated GPU) located at index 0, you can add it like this:

```
lxc config device add <instance_name> <device_name> gpu gputype=physical
id=nvidia.com/igpu=0
```

For a complete example on how to use a GPU CDI pass-through, see *How to pass an NVIDIA GPU to a container* (page 144).

gputype: mdev

> **Note**
>
> The `mdev` GPU type is supported only for VMs. It does not support hotplugging.

An `mdev` GPU device creates and passes a virtual GPU (vGPU) through into the instance. You can check the list of available `mdev` profiles by running *`lxc info --resources`* (page 782).

Device options

GPU devices of type `mdev` have the following device options: `id` DRM card ID of the GPU device (page 494)

Key:	`id`
Type:	string

`mdev` The `mdev` profile to use (page 494)

[246] https://github.com/cncf-tags/container-device-interface

Key:	`mdev`
Type:	string
Default:	`0`
Required:	yes

For example: `i915-GVTg_V5_4`

`pci` PCI address of the GPU device (page 495)

Key:	`pci`
Type:	string

`productid` Product ID of the GPU device (page 495)

Key:	`productid`
Type:	string

`vendorid` Vendor ID of the GPU device (page 495)

Key:	`vendorid`
Type:	string

Configuration examples

Add an `mdev` GPU device to an instance by specifying its `mdev` profile and the PCI address of the GPU:

```
lxc config device add <instance_name> <device_name> gpu gputype=mdev mdev=<mdev_
profile> pci=<pci_address>
```

See *Configure devices* (page 87) for more information.

`gputype: mig`

Note

The `mig` GPU type is supported only for containers. It does not support hotplugging.

A `mig` GPU device creates and passes a MIG compute instance through into the instance. Currently, this requires NVIDIA MIG instances to be pre-created.

Device options

GPU devices of type `mig` have the following device options: `id` DRM card ID of the GPU device (page 495)

Key:	id
Type:	string

`mig.ci` Existing MIG compute instance ID (page 496)

Key:	mig.ci
Type:	integer

`mig.gi` Existing MIG GPU instance ID (page 496)

Key:	mig.gi
Type:	integer

`mig.uuid` Existing MIG device UUID (page 496)

Key:	mig.uuid
Type:	string

You can omit the `MIG-` prefix when specifying this option.

`pci` PCI address of the GPU device (page 496)

Key:	pci
Type:	string

`productid` Product ID of the GPU device (page 496)

Key:	productid
Type:	string

`vendorid` Vendor ID of the GPU device (page 496)

Key:	vendorid
Type:	string

You must set either *`mig.uuid`* (page 496) (NVIDIA drivers 470+) or both *`mig.ci`* (page 496) and *`mig.gi`* (page 496) (old NVIDIA drivers).

Configuration examples

Add a `mig` GPU device to an instance by specifying its UUID and the PCI address of the GPU:

```
lxc config device add <instance_name> <device_name> gpu gputype=mig mig.uuid=<mig_uuid> pci=<pci_address>
```

See *Configure devices* (page 87) for more information.

`gputype: sriov`

> **Note**
>
> The `sriov` GPU type is supported only for VMs. It does not support hotplugging.

An `sriov` GPU device passes a virtual function of an SR-IOV-enabled GPU into the instance.

Device options

GPU devices of type `sriov` have the following device options: `id` DRM card ID of the parent GPU device (page 497)

Key:	`id`
Type:	string

`pci` PCI address of the parent GPU device (page 497)

Key:	`pci`
Type:	string

`productid` Product ID of the parent GPU device (page 497)

Key:	`productid`
Type:	string

`vendorid` Vendor ID of the parent GPU device (page 497)

Key:	`vendorid`
Type:	string

Configuration examples

Add a `sriov` GPU device to an instance by specifying the PCI address of the parent GPU:

```
lxc config device add <instance_name> <device_name> gpu gputype=sriov pci=<pci_
address>
```

See *Configure devices* (page 87) for more information.

Related topics

- *How to pass an NVIDIA GPU to a container* (page 144)
- *Why does my VM stop responding when I try to pass through a GPU?* (page 334)

Type: `infiniband`

> **Note**
>
> The `infiniband` device type is supported for both containers and VMs. It supports hot-plugging only for containers, not for VMs.

LXD supports two different kinds of network types for InfiniBand devices:

- `physical`: Passes a physical device from the host through to the instance. The targeted device will vanish from the host and appear in the instance.
- `sriov`: Passes a virtual function of an SR-IOV-enabled physical network device into the instance.

  > **Note**
  >
  > InfiniBand devices support SR-IOV, but in contrast to other SR-IOV-enabled devices, InfiniBand does not support dynamic device creation in SR-IOV mode. Therefore, you must pre-configure the number of virtual functions by configuring the corresponding kernel module.

Device options

`infiniband` devices have the following device options: `hwaddr` MAC address of the new interface (page 498)

Key:	`hwaddr`
Type:	string
Default:	randomly assigned
Required:	no

You can specify either the full 20-byte variant or the short 8-byte variant (which will modify only the last 8 bytes of the parent device).

`mtu` MTU of the new interface (page 498)

Key:	`mtu`
Type:	integer
Default:	parent MTU
Required:	no

`name` Name of the interface inside the instance (page 498)

Key:	`name`
Type:	string
Default:	kernel assigned
Required:	no

`nictype` Device type (page 499)

Key:	`nictype`
Type:	string
Required:	yes

Possible values are `physical` and `sriov`.

`parent` The name of the host device or bridge (page 499)

Key:	`parent`
Type:	string
Required:	yes

Configuration examples

Add a `physical` `infiniband` device to an instance:

```
lxc config device add <instance_name> <device_name> infiniband nictype=physical
parent=<device>
```

Add an `sriov` `infiniband` device to an instance:

```
lxc config device add <instance_name> <device_name> infiniband nictype=sriov
parent=<sriov_enabled_device>
```

See *Configure devices* (page 87) for more information.

Type: `proxy`

Note

The `proxy` device type is supported for both containers (NAT and non-NAT modes) and VMs (NAT mode only). It supports hotplugging for both containers and VMs.

Proxy devices allow you to forward network connections between a host and an instance running on that host.

You can use them to:

- Forward traffic from an address on the host to an address inside the instance.
- Do the reverse, enabling an address inside the instance to connect through the host.

In *NAT mode* (page 500), proxy devices support TCP and UDP proxying (traffic forwarding). In non-NAT mode, proxy devices can also forward traffic between Unix sockets, which is useful for tasks such as forwarding a GUI or audio traffic from a container to the host system. Additionally, they can proxy traffic across different protocols—for example, forwarding traffic from a TCP listener on the host to a Unix socket inside a container.

The supported connection types are:

- `tcp <-> tcp`
- `udp <-> udp`
- `unix <-> unix`
- `tcp <-> unix`
- `unix <-> tcp`
- `tcp <-> udp`
- `unix <-> udp`

To add a `proxy` device, use the following command:

```
lxc config device add <instance_name> <device_name> proxy listen=<type>:<addr>:
<port>[-<port>][,<port>] connect=<type>:<addr>:<port> bind=<host/instance_name>
```

> **Tip**
>
> Using a proxy device in NAT mode is very similar to adding a *network forward* (page 235).
>
> The difference is that network forwards are applied on a network level, while a proxy device is added for an instance. In addition, network forwards cannot be used to proxy traffic between different connection types.

NAT mode

The proxy device supports a NAT mode (`nat=true`), which forwards packets using NAT instead of creating a separate proxy connection.

This mode has the benefit that the client address is maintained without requiring the target destination to support the HAProxy PROXY protocol. This is necessary for passing client addresses in non-NAT mode.

However, NAT mode is only available when the host running the instance also acts as the gateway. This is the typical case when using `lxdbr0`, for example.

In NAT mode, the supported connection types are:

- `tcp <-> tcp`
- `udp <-> udp`

When configuring a proxy device with `nat=true`, you must ensure that the target instance has a static IP configured on its NIC device.

Specifying IP addresses

Use the following command to configure a static IP for an instance NIC:

```
lxc config device set <instance_name> <nic_name> ipv4.address=<ipv4_address> ipv6.
address=<ipv6_address>
```

To define a static IPv6 address, the parent managed network must have `ipv6.dhcp.stateful` enabled.

When defining IPv6 addresses, use square bracket notation. Example:

```
connect=tcp:[2001:db8::1]:80
```

You can specify that the connect address should be the IP of the instance by setting the connect IP to the wildcard address, which is `0.0.0.0` for IPv4 and `[::]` for IPv6.

> **Note**
>
> The listen address can also use wildcard addresses in non-NAT mode. However, when using NAT mode, you must specify an IP address on the LXD host.

Device options

`proxy` devices have the following device options: `bind` Which side to bind on (page 501)

Key:	`bind`
Type:	string
Default:	`host`
Required:	no

Possible values are `host` and `instance`.

`connect` Address and port to connect to (page 501)

Key:	`connect`
Type:	string
Required:	yes

Use the following format to specify the address and port: `<type>:<addr>:<port>[-<port>][,<port>]`

`gid` GID of the owner of the listening Unix socket (page 501)

Key:	`gid`
Type:	integer
Default:	`0`
Required:	no

`listen` Address and port to bind and listen (page 501)

Key:	`listen`
Type:	string
Required:	yes

Use the following format to specify the address and port: `<type>:<addr>:<port>[-<port>][,<port>]`

`mode` Mode for the listening Unix socket (page 502)

Key:	`mode`
Type:	integer
Default:	`0644`
Required:	no

`nat` Whether to optimize proxying via NAT (page 502)

Key:	`nat`
Type:	bool
Default:	`false`
Required:	no

This option requires that the instance NIC has a static IP address.

`proxy_protocol` Whether to use the HAProxy PROXY protocol (page 502)

Key:	`proxy_protocol`
Type:	bool
Default:	`false`
Required:	no

This option specifies whether to use the HAProxy PROXY protocol to transmit sender information.

`security.gid` What GID to drop privilege to (page 502)

Key:	`security.gid`
Type:	integer
Default:	`0`
Required:	no

`security.uid` What UID to drop privilege to (page 502)

Key:	`security.uid`
Type:	integer
Default:	`0`
Required:	no

`uid` UID of the owner of the listening Unix socket (page 503)

Key:	`uid`
Type:	integer
Default:	`0`
Required:	no

Configuration examples

Add a `proxy` device that forwards traffic from one address (the `listen` address) to another address (the `connect` address) using NAT mode:

```
lxc config device add <instance_name> <device_name> proxy nat=true listen=tcp:<ip_
address>:<port> connect=tcp:<ip_address>:<port>
```

Add a `proxy` device that forwards traffic going to a specific IP to a Unix socket on an instance that might not have a network connection:

```
lxc config device add <instance_name> <device_name> proxy listen=tcp:<ip_address>:
<port> connect=unix:/<socket_path_on_instance>
```

Add a `proxy` device that forwards traffic going to a Unix socket on an instance that might not have a network connection to a specific IP address:

```
lxc config device add <instance_name> <device_name> proxy bind=instance
listen=unix:/<socket_path_on_instance> connect=tcp:<ip_address>:<port>
```

See *Configure devices* (page 87) for more information.

Type: `unix-hotplug`

> **Note**
>
> The `unix-hotplug` device type is supported for containers. It supports hotplugging.

Unix hotplug devices make the requested Unix device appear as a device in the container (under `/dev`). If the device exists on the host system, you can read from it and write to it.

The implementation depends on `systemd-udev` to be run on the host.

Device options

`unix-hotplug` devices have the following device options: `gid` GID of the device owner in the container (page 503)

Key:	`gid`
Type:	integer
Default:	`0`

`mode` Mode of the device in the container (page 503)

Key:	`mode`
Type:	integer
Default:	`0660`

`ownership.inherit` Whether this device inherits ownership (GID and/or UID) from the host (page 504)

Key:	`ownership.inherit`
Type:	bool
Default:	`false`

`productid` Product ID of the Unix device (page 504)

Key:	`productid`
Type:	string

`required` Whether this device is required to start the container (page 504)

Key:	`required`
Type:	bool
Default:	`false`

The default is `false`, which means that all devices can be hotplugged.

`subsystem` Subsystem of the Unix device (page 504)

Key:	`subsystem`
Type:	string

`uid` UID of the device owner in the container (page 504)

Key:	`uid`
Type:	integer
Default:	`0`

`vendorid` Vendor ID of the Unix device (page 504)

Key:	`vendorid`
Type:	string

Configuration examples

Add a `unix-hotplug` device to an instance by specifying its vendor ID, product ID, and/or subsystem:

```
lxc config device add <instance_name> <device_name> unix-hotplug vendorid=<vendor_
ID> productid=<product_ID> subsystem=<subsystem>
```

See *Configure devices* (page 87) for more information.

Type: `tpm`

> **Note**
>
> The `tpm` device type is supported for both containers and VMs. It supports hotplugging only for containers, not for VMs.

TPM devices enable access to a TPM (Trusted Platform Module) emulator.

TPM devices can be used to validate the boot process and ensure that no steps in the boot chain have been tampered with, and they can securely generate and store encryption keys.

LXD uses a software TPM that supports TPM 2.0. For containers, the main use case is sealing certificates, which means that the keys are stored outside of the container, making it virtually impossible for attackers to retrieve them. For virtual machines, TPM can be used both for sealing certificates and for validating the boot process, which allows using full disk encryption compatible with, for example, Windows BitLocker.

Device options

`tpm` devices have the following device options: `path` Path inside the container (page 505)

Key:	`path`
Type:	string
Required:	for containers

For example: `/dev/tpm0`

`pathrm` Resource manager path inside the container (page 505)

Key:	`pathrm`
Type:	string
Required:	for containers

For example: `/dev/tpmrm0`

Configuration examples

Add a `tpm` device to a container by specifying its path and the resource manager path:

```
lxc config device add <instance_name> <device_name> tpm path=<path_on_instance>
pathrm=<resource_manager_path>
```

Add a `tpm` device to a virtual machine:

```
lxc config device add <instance_name> <device_name> tpm
```

See *Configure devices* (page 87) for more information.

Type: `pci`

> **Note**
>
> The `pci` device type is supported for VMs. It does not support hotplugging.

PCI devices are used to pass raw PCI devices from the host into a virtual machine.

They are mainly intended to be used for specialized single-function PCI cards like sound cards or video capture cards. In theory, you can also use them for more advanced PCI devices like GPUs or network cards, but it's usually more convenient to use the specific device types that LXD provides for these devices (*gpu device* (page 491) or *nic device* (page 449)).

Device options

`pci` devices have the following device options: `address` PCI address of the device (page 506)

Key:	`address`
Type:	string
Required:	yes

Configuration examples

Add a `pci` device to a virtual machine by specifying its PCI address:

```
lxc config device add <instance_name> <device_name> pci address=<pci_address>
```

To determine the PCI address, you can use **`lspci`**, for example.

See *Configure devices* (page 87) for more information.

Units for storage and network limits

Any value that represents bytes or bits can make use of a number of suffixes to make it easier to understand what a particular limit is.

Both decimal and binary (kibi) units are supported, with the latter mostly making sense for storage limits.

The full list of bit suffixes currently supported is:

- bit (1)
- kbit (1000)
- Mbit (1000^2)
- Gbit (1000^3)
- Tbit (1000^4)
- Pbit (1000^5)
- Ebit (1000^6)
- Kibit (1024)
- Mibit (1024^2)
- Gibit (1024^3)
- Tibit (1024^4)
- Pibit (1024^5)
- Eibit (1024^6)

The full list of byte suffixes currently supported is:

- B or bytes (1)
- kB (1000)
- MB (1000^2)
- GB (1000^3)
- TB (1000^4)
- PB (1000^5)
- EB (1000^6)
- KiB (1024)
- MiB (1024^2)
- GiB (1024^3)
- TiB (1024^4)
- PiB (1024^5)
- EiB (1024^6)

Related topics

How-to guides:

- *Instances* (page 73)

Explanation:

- *Instance types in LXD* (page 347)

4.2.4. Preseed YAML file fields

You can configure a new LXD installation and reconfigure an existing installation with a preseed YAML file.

The preseed YAML file fields are as follows:

```yaml
config:
  core.https_address: ""
  images.auto_update_interval: 6

networks:
  - config:
      ipv4.address: auto
      ipv4.nat: "true"
      ipv6.address: auto
      ipv6.nat: "true"
    description: ""
    name: lxdbr0
    type: bridge
    project: default

storage_pools:
  - config: {}
    description: ""
    name: default
    driver: zfs

storage_volumes:
- name: my-vol
  pool: data

profiles:
  - config:
      limits.memory: 2GiB
    description: Default LXD profile
    devices:
      eth0:
        name: eth0
        network: lxdbr0
        type: nic
      root:
        path: /
        pool: default
        type: disk
    name: default

projects:
  - config:
      features.images: "true"
      features.networks: "true"
      features.networks.zones: "true"
```

(continues on next page)

(continued from previous page)

```yaml
      features.profiles: "true"
      features.storage.buckets: "true"
      features.storage.volumes: "true"
    description: Default LXD project
    name: default

cluster:
  enabled: true
  server_address: ""
  cluster_token: ""
  member_config:
  - entity: storage-pool
    name: default
    key: source
    value: ""
  - entity: storage-pool
    name: my-pool
    key: source
    value: ""
  - entity: storage-pool
    name: my-pool
    key: driver
    value: "zfs"
```

Related topics

How-to guides:

- *How to initialize LXD* (page 35)

4.2.5. Project configuration

Projects can be configured through a set of key/value configuration options. See *Configure a project* (page 164) for instructions on how to set these options.

The key/value configuration is namespaced. The following options are available:

- *Project features* (page 509)
- *Project limits* (page 511)
- *Project restrictions* (page 513)
- *Project-specific configuration* (page 520)

Project features

The project features define which entities are isolated in the project and which are inherited from the `default` project.

If a `feature.*` option is set to `true`, the corresponding entity is isolated in the project.

Note

When you create a project without explicitly configuring a specific option, this option is set to the initial value given in the following table.

However, if you unset one of the `feature.*` options, it does not go back to the initial value, but to the default value. The default value for all `feature.*` options is `false`.

`features.images` Whether to use a separate set of images for the project (page 510)

Key:	`features.images`
Type:	bool
Default:	`false`
Initial value:	`true`

This setting applies to both images and image aliases.

`features.networks` Whether to use a separate set of networks for the project (page 510)

Key:	`features.networks`
Type:	bool
Default:	`false`
Initial value:	`false`

`features.networks.zones` Whether to use a separate set of network zones for the project (page 510)

Key:	`features.networks.zones`
Type:	bool
Default:	`false`
Initial value:	`false`

`features.profiles` Whether to use a separate set of profiles for the project (page 510)

Key:	`features.profiles`
Type:	bool
Default:	`false`
Initial value:	`true`

`features.storage.buckets` Whether to use a separate set of storage buckets for the project (page 510)

Key:	`features.storage.buckets`
Type:	bool
Default:	`false`
Initial value:	`true`

`features.storage.volumes` Whether to use a separate set of storage volumes for the project (page 510)

Key:	`features.storage.volumes`
Type:	bool
Default:	`false`
Initial value:	`true`

Project limits

Project limits define a hard upper bound for the resources that can be used by the containers and VMs that belong to a project.

Depending on the `limits.*` option, the limit applies to the number of entities that are allowed in the project (for example, *`limits.containers`* (page 511) or *`limits.networks`* (page 512)) or to the aggregate value of resource usage for all instances in the project (for example, *`limits.cpu`* (page 511) or *`limits.processes`* (page 513)). In the latter case, the limit usually applies to the *Resource limits* (page 421) that are configured for each instance (either directly or via a profile), and not to the resources that are actually in use.

For example, if you set the project's *`limits.memory`* (page 512) configuration to `50GiB`, the sum of the individual values of all *`limits.memory`* (page 423) configuration keys defined on the project's instances will be kept under 50 GiB.

Similarly, setting the project's *`limits.cpu`* (page 511) configuration key to `100` means that the sum of individual *`limits.cpu`* (page 421) values will be kept below 100.

When using project limits, the following conditions must be fulfilled:

- When you set one of the `limits.*` configurations and there is a corresponding configuration for the instance, all instances in the project must have the corresponding configuration defined (either directly or via a profile). See *Resource limits* (page 421) for the instance configuration options.
- The *`limits.cpu`* (page 511) configuration cannot be used if *CPU pinning* (page 426) is enabled. This means that to use *`limits.cpu`* (page 511) on a project, the *`limits.cpu`* (page 421) configuration of each instance in the project must be set to a number of CPUs, not a set or a range of CPUs.
- The *`limits.memory`* (page 512) configuration must be set to an absolute value, not a percentage.

`limits.containers` Maximum number of containers that can be created in the project (page 511)

Key:	`limits.containers`
Type:	integer

`limits.cpu` Maximum number of CPUs to use in the project (page 511)

Key:	`limits.cpu`
Type:	integer

This value is the maximum value for the sum of the individual *limits.cpu* (page 421) configurations set on the instances of the project.

`limits.disk` Maximum disk space used by the project (page 512)

Key:	`limits.disk`
Type:	string

This value is the maximum value of the aggregate disk space used by all instance volumes, custom volumes, and images of the project.

`limits.disk.pool.POOL_NAME` Maximum disk space used by the project on this pool (page 512)

Key:	`limits.disk.pool.POOL_NAME`
Type:	string

This value is the maximum value of the aggregate disk space used by all instance volumes, custom volumes, and images of the project on this specific storage pool.

When set to 0, the pool is excluded from storage pool list for the project.

`limits.instances` Maximum number of instances that can be created in the project (page 512)

Key:	`limits.instances`
Type:	integer

`limits.memory` Usage limit for the host's memory for the project (page 512)

Key:	`limits.memory`
Type:	string

The value is the maximum value for the sum of the individual *limits.memory* (page 423) configurations set on the instances of the project.

`limits.networks` Maximum number of networks that the project can have (page 512)

Key:	`limits.networks`
Type:	integer

limits.networks.uplink_ips.ipv4.NETWORK_NAME Quota of IPv4 addresses from a specified uplink network that can be used by entities in this project (page 512)

Key:	limits.networks.uplink_ips.ipv4.NETWORK_NAME
Type:	string

Maximum number of IPv4 addresses that this project can consume from the specified uplink network. This number of IPs can be consumed by networks, forwards and load balancers in this project.

limits.networks.uplink_ips.ipv6.NETWORK_NAME Quota of IPv6 addresses from a specified uplink network that can be used by entities in this project (page 513)

Key:	limits.networks.uplink_ips.ipv6.NETWORK_NAME
Type:	string

Maximum number of IPv6 addresses that this project can consume from the specified uplink network. This number of IPs can be consumed by networks, forwards and load balancers in this project.

limits.processes Maximum number of processes within the project (page 513)

Key:	limits.processes
Type:	integer

This value is the maximum value for the sum of the individual *limits.processes* (page 425) configurations set on the instances of the project.

limits.virtual-machines Maximum number of VMs that can be created in the project (page 513)

Key:	limits.virtual-machines
Type:	integer

Project restrictions

To prevent the instances of a project from accessing security-sensitive features (such as container nesting or raw LXC configuration), set the *restricted* (page 514) configuration option to true. You can then use the various restricted.* options to pick individual features that would normally be blocked by *restricted* (page 514) and allow them, so they can be used by the instances of the project.

For example, to restrict a project and block all security-sensitive features, but allow container nesting, enter the following commands:

```
lxc project set <project_name> restricted=true
lxc project set <project_name> restricted.containers.nesting=allow
```

Each security-sensitive feature has an associated `restricted.*` project configuration option. If you want to allow the usage of a feature, change the value of its `restricted.*` option. Most `restricted.*` configurations are binary switches that can be set to either `block` (the default) or `allow`. However, some options support other values for more fine-grained control.

> **Note**
>
> You must set the `restricted` configuration to `true` for any of the `restricted.*` options to be effective. If `restricted` is set to `false`, changing a `restricted.*` option has no effect.
>
> Setting all `restricted.*` keys to `allow` is equivalent to setting `restricted` itself to `false`.

`restricted` Whether to block access to security-sensitive features (page 514)

Key:	`restricted`
Type:	bool
Default:	`false`

This option must be enabled to allow the `restricted.*` keys to take effect. To temporarily remove the restrictions, you can disable this option instead of clearing the related keys.

`restricted.backups` Whether to prevent creating instance or volume backups (page 514)

Key:	`restricted.backups`
Type:	string
Default:	`block`

Possible values are `allow` or `block`.

`restricted.cluster.groups` Cluster groups that can be targeted (page 514)

Key:	`restricted.cluster.groups`
Type:	string

If specified, this option prevents targeting cluster groups other than the provided ones.

`restricted.cluster.target` Whether to prevent targeting of cluster members (page 514)

Key:	`restricted.cluster.target`
Type:	string
Default:	`block`

Possible values are allow or block. When set to allow, this option allows targeting of cluster members (either directly or via a group) when creating or moving instances.

restricted.containers.interception Whether to prevent using system call interception options (page 515)

Key:	restricted.containers.interception
Type:	string
Default:	block

Possible values are allow, block, or full. When set to allow, interception options that are usually safe are allowed. File system mounting remains blocked.

restricted.containers.lowlevel Whether to prevent using low-level container options (page 515)

Key:	restricted.containers.lowlevel
Type:	string
Default:	block

Possible values are allow or block. When set to allow, low-level container options like *raw.lxc* (page 431), *raw.idmap* (page 431), volatile.*, etc. can be used.

restricted.containers.nesting Whether to prevent running nested LXD (page 515)

Key:	restricted.containers.nesting
Type:	string
Default:	block

Possible values are allow or block. When set to allow, *security.nesting* (page 436) can be set to true for an instance.

restricted.containers.privilege Which settings for privileged containers to prevent (page 515)

Key:	restricted.containers.privilege
Type:	string
Default:	unprivileged

Possible values are unprivileged, isolated, and allow.

- When set to unpriviliged, this option prevents setting *security.privileged* (page 436) to true.
- When set to isolated, this option prevents setting *security.privileged* (page 436) to true and forces using a unique idmap per container using *security.idmap.isolated* (page 436) set to true.

- When set to `allow`, there is no restriction.

`restricted.devices.disk` Which disk devices can be used (page 516)

Key:	`restricted.devices.disk`
Type:	string
Default:	`managed`

Possible values are `allow`, `block`, or `managed`.

- When set to `block`, this option prevents using all disk devices except the root one.
- When set to `managed`, this option allows using disk devices only if `pool=` is set.
- When set to `allow`, there is no restriction on which disk devices can be used.

> **Important**
>
> When allowing all disk devices, make sure to set *`restricted.devices.disk.paths`* (page 516) to a list of path prefixes that you want to allow. If you do not restrict the allowed paths, users can attach any disk device, including shifted devices (`disk` devices with *`shift`* (page 480) set to `true`), which can be used to gain root access to the system.

`restricted.devices.disk.paths` Which `source` can be used for `disk` devices (page 516)

Key:	`restricted.devices.disk.paths`
Type:	string

If *`restricted.devices.disk`* (page 516) is set to `allow`, this option controls which `source` can be used for `disk` devices. Specify a comma-separated list of path prefixes that restrict the `source` setting. If this option is left empty, all paths are allowed.

`restricted.devices.gpu` Whether to prevent using devices of type `gpu` (page 516)

Key:	`restricted.devices.gpu`
Type:	string
Default:	`block`

Possible values are `allow` or `block`.

`restricted.devices.infiniband` Whether to prevent using devices of type `infiniband` (page 516)

Key:	`restricted.devices.infiniband`
Type:	string
Default:	`block`

Possible values are `allow` or `block`.

`restricted.devices.nic` Which network devices can be used (page 517)

Key:	`restricted.devices.nic`
Type:	string
Default:	`managed`

Possible values are `allow`, `block`, or `managed`.

- When set to `block`, this option prevents using all network devices.
- When set to `managed`, this option allows using network devices only if `network=` is set.
- When set to `allow`, there is no restriction on which network devices can be used.

`restricted.devices.pci` Whether to prevent using devices of type `pci` (page 517)

Key:	`restricted.devices.pci`
Type:	string
Default:	`block`

Possible values are `allow` or `block`.

`restricted.devices.proxy` Whether to prevent using devices of type `proxy` (page 517)

Key:	`restricted.devices.proxy`
Type:	string
Default:	`block`

Possible values are `allow` or `block`.

`restricted.devices.unix-block` Whether to prevent using devices of type `unix-block` (page 517)

Key:	`restricted.devices.unix-block`
Type:	string
Default:	`block`

Possible values are `allow` or `block`.

`restricted.devices.unix-char` Whether to prevent using devices of type `unix-char` (page 517)

Key:	`restricted.devices.unix-char`
Type:	string
Default:	`block`

Possible values are `allow` or `block`.

`restricted.devices.unix-hotplug` Whether to prevent using devices of type `unix-hotplug` (page 518)

Key:	`restricted.devices.unix-hotplug`
Type:	string
Default:	`block`

Possible values are `allow` or `block`.

`restricted.devices.usb` Whether to prevent using devices of type `usb` (page 518)

Key:	`restricted.devices.usb`
Type:	string
Default:	`block`

Possible values are `allow` or `block`.

`restricted.idmap.gid` Which host GID ranges are allowed in `raw.idmap` (page 518)

Key:	`restricted.idmap.gid`
Type:	string

This option specifies the host GID ranges that are allowed in the instance's *`raw.idmap`* (page 431) setting.

`restricted.idmap.uid` Which host UID ranges are allowed in `raw.idmap` (page 518)

Key:	`restricted.idmap.uid`
Type:	string

This option specifies the host UID ranges that are allowed in the instance's *`raw.idmap`* (page 431) setting.

`restricted.networks.access` Which network names are allowed for use in this project (page 518)

Key:	`restricted.networks.access`
Type:	string

Specify a comma-delimited list of network names that are allowed for use in this project. If this option is not set, all networks are accessible.

Note that this setting depends on the *restricted.devices.nic* (page 517) setting.

`restricted.networks.subnets` Which network subnets are allocated for use in this project (page 519)

Key:	`restricted.networks.subnets`
Type:	string
Default:	`block`

Specify a comma-delimited list of CIDR network routes from the uplink network's *ipv4.routes* (page 597) *ipv6.routes* (page 598) that are allowed for use in this project. Use the form `<uplink>:<subnet>`.

Example value: `lxdbr0:192.0.168.0/24,lxdbr0:10.1.19.5/32`

`restricted.networks.uplinks` Which network names can be used as uplink in this project (page 519)

Key:	`restricted.networks.uplinks`
Type:	string
Default:	`block`

Specify a comma-delimited list of network names that can be used as uplink for networks in this project.

`restricted.networks.zones` Which network zones can be used in this project (page 519)

Key:	`restricted.networks.zones`
Type:	string
Default:	`block`

Specify a comma-delimited list of network zones that can be used (or something under them) in this project.

`restricted.snapshots` Whether to prevent creating instance or volume snapshots (page 519)

Key:	restricted.snapshots
Type:	string
Default:	block

restricted.virtual-machines.lowlevel Whether to prevent using low-level VM options (page 520)

Key:	restricted.virtual-machines.lowlevel
Type:	string
Default:	block

Possible values are allow or block. When set to allow, low-level VM options like *raw.qemu* (page 431), volatile.*, etc. can be used.

Project-specific configuration

There are some *Server configuration* (page 401) options that you can override for a project. In addition, you can add user metadata for a project. backups.compression_algorithm Compression algorithm to use for backups (page 520)

Key:	backups.compression_algorithm
Type:	string

Specify which compression algorithm to use for backups in this project. Possible values are bzip2, gzip, lzma, xz, or none.

images.auto_update_cached Whether to automatically update cached images in the project (page 520)

Key:	images.auto_update_cached
Type:	bool

images.auto_update_interval Interval at which to look for updates to cached images (page 520)

Key:	images.auto_update_interval
Type:	integer

Specify the interval in hours. To disable looking for updates to cached images, set this option to 0.

images.compression_algorithm Compression algorithm to use for new images in the project (page 520)

Key:	`images.compression_algorithm`
Type:	string

Possible values are `bzip2`, `gzip`, `lzma`, `xz`, or `none`.

`images.default_architecture` Default architecture to use in a mixed-architecture cluster (page 521)

Key:	`images.default_architecture`
Type:	string

`images.remote_cache_expiry` When an unused cached remote image is flushed in the project (page 521)

Key:	`images.remote_cache_expiry`
Type:	integer

Specify the number of days after which the unused cached image expires.

`user.*` User-provided free-form key/value pairs (page 521)

Key:	`user.*`
Type:	string

Related topics

How-to guides:

- *Projects* (page 161)

Explanation:

- *Instances grouping with projects* (page 368)

4.2.6. Storage drivers

LXD supports the following storage drivers for storing images, instances and custom volumes:

Btrfs - `btrfs`

Btrfs (B-tree file system) is a local file system based on the COW (copy-on-write) principle. COW means that data is stored to a different block after it has been modified instead of overwriting the existing data, reducing the risk of data corruption. Unlike other file systems, Btrfs is extent-based, which means that it stores data in contiguous areas of memory.

In addition to basic file system features, Btrfs offers RAID and volume management, pooling, snapshots, checksums, compression and other features.

To use Btrfs, make sure you have `btrfs-progs` installed on your machine.

Terminology

A Btrfs file system can have *subvolumes*, which are named binary subtrees of the main tree of the file system with their own independent file and directory hierarchy. A *Btrfs snapshot* is a special type of subvolume that captures a specific state of another subvolume. Snapshots can be read-write or read-only.

`btrfs` driver in LXD

The `btrfs` driver in LXD uses a subvolume per instance, image and snapshot. When creating a new entity (for example, launching a new instance), it creates a Btrfs snapshot.

Btrfs doesn't natively support storing block devices. Therefore, when using Btrfs for VMs, LXD creates a big file on disk to store the VM. This approach is not very efficient and might cause issues when creating snapshots.

Btrfs can be used as a storage backend inside a container in a nested LXD environment. In this case, the parent container itself must use Btrfs. Note, however, that the nested LXD setup does not inherit the Btrfs quotas from the parent (see *Quotas* (page 522) below).

Quotas

Btrfs supports storage quotas via qgroups. Btrfs qgroups are hierarchical, but new subvolumes will not automatically be added to the qgroups of their parent subvolumes. This means that users can trivially escape any quotas that are set. Therefore, if strict quotas are needed, you should consider using a different storage driver (for example, ZFS with `refquota` or LVM with Btrfs on top).

When using quotas, you must take into account that Btrfs extents are immutable. When blocks are written, they end up in new extents. The old extents remain until all their data is dereferenced or rewritten. This means that a quota can be reached even if the total amount of space used by the current files in the subvolume is smaller than the quota.

Note

This issue is seen most often when using VMs on Btrfs, due to the random I/O nature of using raw disk image files on top of a Btrfs subvolume.

Therefore, you should never use VMs with Btrfs storage pools.

If you really need to use VMs with Btrfs storage pools, set the instance root disk's *`size.state`* (page 484) property to twice the size of the root disk's size. This configuration allows all blocks in the disk image file to be rewritten without reaching the qgroup quota. Setting the *`btrfs.mount_options`* (page 523) storage pool option to `compress-force` can also avoid this scenario, because a side effect of enabling compression is to reduce the maximum extent size such that block rewrites don't cause as much storage to be double-tracked. However, this is a storage pool option, and it therefore affects all volumes on the pool.

Configuration options

The following configuration options are available for storage pools that use the `btrfs` driver and for storage volumes in these pools.

Storage pool configuration

`btrfs.mount_options` Mount options for block devices (page 523)

Key:	`btrfs.mount_options`
Type:	string
Default:	`user_subvol_rm_allowed`
Scope:	global

`size` Size of the storage pool (for loop-based pools) (page 523)

Key:	`size`
Type:	string
Default:	auto (20% of free disk space, >= 5 GiB and <= 30 GiB)
Scope:	local

When creating loop-based pools, specify the size in bytes (*suffixes* (page 506) are supported). You can increase the size to grow the storage pool.

The default (`auto`) creates a storage pool that uses 20% of the free disk space, with a minimum of 5 GiB and a maximum of 30 GiB.

`source` Path to an existing block device, loop file, or Btrfs subvolume (page 523)

Key:	`source`
Type:	string
Scope:	local

`source.wipe` Whether to wipe the block device before creating the pool (page 523)

Key:	`source.wipe`
Type:	bool
Default:	`false`
Scope:	local

Set this option to `true` to wipe the block device specified in `source` prior to creating the storage pool.

> **Tip**
>
> In addition to these configurations, you can also set default values for the storage volume configurations. See *Configure default values for storage volumes* (page 190).

Storage volume configuration

`security.shared` Enable volume sharing (page 524)

Key:	`security.shared`
Type:	bool
Default:	same as `volume.security.shared` or `false`
Condition:	virtual-machine or custom block volume
Scope:	global

Enabling this option allows sharing the volume across multiple instances despite the possibility of data loss.

`security.shifted` Enable ID shifting overlay (page 524)

Key:	`security.shifted`
Type:	bool
Default:	same as `volume.security.shifted` or `false`
Condition:	custom volume
Scope:	global

Enabling this option allows attaching the volume to multiple isolated instances.

`security.unmapped` Disable ID mapping for the volume (page 524)

Key:	`security.unmapped`
Type:	bool
Default:	same as `volume.security.unmappped` or `false`
Condition:	custom volume
Scope:	global

`size` Size/quota of the storage volume (page 524)

Key:	`size`
Type:	string
Default:	same as `volume.size`
Condition:	appropriate driver
Scope:	global

`snapshots.expiry` When snapshots are to be deleted (page 524)

Key:	`snapshots.expiry`
Type:	string
Default:	same as `volume.snapshots.expiry`
Condition:	custom volume
Scope:	global

Specify an expression like `1M 2H 3d 4w 5m 6y`.

`snapshots.pattern` Template for the snapshot name (page 524)

Key:	`snapshots.pattern`
Type:	string
Default:	same as `volume.snapshots.pattern` or `snap%d`
Condition:	custom volume
Scope:	global

You can specify a naming template that is used for scheduled snapshots and unnamed snapshots.

The `snapshots.pattern` option takes a Pongo2 template string to format the snapshot name.

To add a time stamp to the snapshot name, use the Pongo2 context variable `creation_date`. Make sure to format the date in your template string to avoid forbidden characters in the snapshot name. For example, set `snapshots.pattern` to `{{ creation_date|date:'2006-01-02_15-04-05' }}` to name the snapshots after their time of creation, down to the precision of a second.

Another way to avoid name collisions is to use the placeholder `%d` in the pattern. For the first snapshot, the placeholder is replaced with `0`. For subsequent snapshots, the existing snapshot names are taken into account to find the highest number at the placeholder's position. This number is then incremented by one for the new name.

`snapshots.schedule` Schedule for automatic volume snapshots (page 525)

Key:	`snapshots.schedule`
Type:	string
Default:	same as `snapshots.schedule`
Condition:	custom volume
Scope:	global

Specify either a cron expression (`<minute> <hour> <dom> <month> <dow>`), a comma-separated list of schedule aliases (`@hourly`, `@daily`, `@midnight`, `@weekly`, `@monthly`, `@annually`, `@yearly`), or leave empty to disable automatic snapshots (the default).

`volatile.idmap.last` JSON-serialized UID/GID map that has been applied to the volume (page 525)

Key:	`volatile.idmap.last`
Type:	string
Condition:	filesystem

`volatile.idmap.next` JSON-serialized UID/GID map that has been applied to the volume (page 525)

Key:	`volatile.idmap.next`
Type:	string
Condition:	filesystem

`volatile.uuid` The volume's UUID (page 526)

Key:	`volatile.uuid`
Type:	string
Default:	random UUID
Scope:	global

Storage bucket configuration

To enable storage buckets for local storage pool drivers and allow applications to access the buckets via the S3 protocol, you must configure the *core.storage_buckets_address* (page 404) server setting. `size` Size/quota of the storage bucket (page 526)

Key:	`size`
Type:	string
Default:	same as `volume.size`
Condition:	appropriate driver
Scope:	local

CephFS - `cephfs`

Ceph[247] is an open-source storage platform that stores its data in a storage cluster based on RADOS (Reliable Autonomic Distributed Object Store). It is highly scalable and, as a distributed system without a single point of failure, very reliable.

> **Tip**
>
> If you want to quickly set up a basic Ceph cluster, check out MicroCeph[248].

Ceph provides different components for block storage and for file systems.

CephFS (Ceph File System) is Ceph's file system component that provides a robust, fully-featured POSIX-compliant distributed file system. Internally, it maps files to Ceph objects and stores file metadata (for example, file ownership, directory paths, access permissions) in a separate data pool.

Terminology

Ceph uses the term *object* for the data that it stores. The daemon that is responsible for storing and managing data is the *Ceph* OSD *(Object Storage Daemon)*. Ceph's storage is divided into *pools*, which are logical partitions for storing objects. They are also referred to as *data pools*, *storage pools* or *OSD pools*.

A *CephFS file system* consists of two OSD storage pools, one for the actual data and one for the file metadata.

[247] https://ceph.io/en/
[248] https://canonical.com/microcloud

Note

The cephfs driver can only be used for custom storage volumes with content type filesystem.

For other storage volumes, use the *Ceph* (page 534) driver. That driver can also be used for custom storage volumes with content type filesystem, but it implements them through Ceph RBD images.

Unlike other storage drivers, this driver does not set up the storage system but assumes that you already have a Ceph cluster installed.

You can either create the CephFS file system that you want to use beforehand and specify it through the *source* (page 529) option, or specify the *cephfs.create_missing* (page 527) option to automatically create the file system and the data and metadata OSD pools (with the names given in *cephfs.data_pool* (page 528) and *cephfs.meta_pool* (page 528)).

This driver also behaves differently than other drivers in that it provides remote storage. As a result and depending on the internal network, storage access might be a bit slower than for local storage. On the other hand, using remote storage has big advantages in a cluster setup, because all cluster members have access to the same storage pools with the exact same contents, without the need to synchronize storage pools.

LXD assumes that it has full control over the OSD storage pool. Therefore, you should never maintain any file system entities that are not owned by LXD in a LXD OSD storage pool, because LXD might delete them.

The cephfs driver in LXD supports snapshots if snapshots are enabled on the server side.

Configuration options

The following configuration options are available for storage pools that use the cephfs driver and for storage volumes in these pools.

Storage pool configuration

cephfs.cluster_name Name of the Ceph cluster that contains the CephFS file system (page 527)

Key:	cephfs.cluster_name
Type:	string
Default:	ceph
Scope:	global

cephfs.create_missing Automatically create the CephFS file system (page 527)

Key:	`cephfs.create_missing`
Type:	bool
Default:	`false`
Scope:	global

Use this option if the CephFS file system does not exist yet. LXD will then automatically create the file system and the missing data and metadata OSD pools.

`cephfs.data_pool` Data OSD pool name (page 528)

Key:	`cephfs.data_pool`
Type:	string
Scope:	global

This option specifies the name for the data OSD pool that should be used when creating a file system automatically.

`cephfs.fscache` Enable use of kernel `fscache` and `cachefilesd` (page 528)

Key:	`cephfs.fscache`
Type:	bool
Default:	`false`
Scope:	global

`cephfs.meta_pool` Metadata OSD pool name (page 528)

Key:	`cephfs.meta_pool`
Type:	string
Scope:	global

This option specifies the name for the file metadata OSD pool that should be used when creating a file system automatically.

`cephfs.osd_pg_num` Number of placement groups when creating missing OSD pools (page 528)

Key:	`cephfs.osd_pg_num`
Type:	string
Scope:	global

This option specifies the number of OSD pool placement groups (`pg_num`) to use when creating a missing OSD pool.

`cephfs.osd_pool_size` Number of RADOS object replicas. Set to 1 for no replication. (page 528)

Key:	`cephfs.osd_pool_size`
Type:	string
Default:	`3`

This option specifies the number of OSD pool replicas to use when creating an OSD pool.

`cephfs.path` The base path for the CephFS mount (page 529)

Key:	`cephfs.path`
Type:	string
Default:	/
Scope:	global

`cephfs.user.name` The Ceph user to use (page 529)

Key:	`cephfs.user.name`
Type:	string
Default:	`admin`
Scope:	global

`source` Existing CephFS file system or file system path to use (page 529)

Key:	`source`
Type:	string
Scope:	local

`volatile.pool.pristine` Whether the CephFS file system was empty on creation time (page 529)

Key:	`volatile.pool.pristine`
Type:	string
Default:	`true`
Scope:	global

> **Tip**
>
> In addition to these configurations, you can also set default values for the storage volume configurations. See *Configure default values for storage volumes* (page 190).

`security.shifted` Enable ID shifting overlay (page 530)

Key:	`security.shifted`
Type:	bool
Default:	same as `volume.security.shifted` or `false`
Condition:	custom volume
Scope:	global

Enabling this option allows attaching the volume to multiple isolated instances.

`security.unmapped` Disable ID mapping for the volume (page 530)

Key:	`security.unmapped`
Type:	bool
Default:	same as `volume.security.unmappped` or `false`
Condition:	custom volume
Scope:	global

`size` Size/quota of the storage volume (page 530)

Key:	`size`
Type:	string
Default:	same as `volume.size`
Condition:	appropriate driver
Scope:	global

`snapshots.expiry` When snapshots are to be deleted (page 530)

Key:	`snapshots.expiry`
Type:	string
Default:	same as `volume.snapshots.expiry`
Condition:	custom volume
Scope:	global

Specify an expression like `1M 2H 3d 4w 5m 6y`.

`snapshots.pattern` Template for the snapshot name (page 530)

Key:	`snapshots.pattern`
Type:	string
Default:	same as `volume.snapshots.pattern` or `snap%d`
Condition:	custom volume
Scope:	global

You can specify a naming template that is used for scheduled snapshots and unnamed snapshots.

The `snapshots.pattern` option takes a Pongo2 template string to format the snapshot name.

To add a time stamp to the snapshot name, use the Pongo2 context variable `creation_date`. Make sure to format the date in your template string to avoid forbidden characters in the snapshot name. For example, set `snapshots.pattern` to `{{ creation_date|date:'2006-01-02_15-04-05' }}` to name the snapshots after their time of creation, down to the precision of a second.

Another way to avoid name collisions is to use the placeholder `%d` in the pattern. For the first snapshot, the placeholder is replaced with `0`. For subsequent snapshots, the existing snapshot names are taken into account to find the highest number at the placeholder's position. This number is then incremented by one for the new name.

`snapshots.schedule` Schedule for automatic volume snapshots (page 531)

Key:	`snapshots.schedule`
Type:	string
Default:	same as `snapshots.schedule`
Condition:	custom volume
Scope:	global

Specify either a cron expression (`<minute> <hour> <dom> <month> <dow>`), a comma-separated list of schedule aliases (`@hourly`, `@daily`, `@midnight`, `@weekly`, `@monthly`, `@annually`, `@yearly`), or leave empty to disable automatic snapshots (the default).

`volatile.idmap.last` JSON-serialized UID/GID map that has been applied to the volume (page 531)

Key:	`volatile.idmap.last`
Type:	string
Condition:	filesystem

`volatile.idmap.next` JSON-serialized UID/GID map that has been applied to the volume (page 531)

Key:	`volatile.idmap.next`
Type:	string
Condition:	filesystem

`volatile.uuid` The volume's UUID (page 531)

Key:	`volatile.uuid`
Type:	string
Default:	random UUID
Scope:	global

Ceph Object - `cephobject`

Ceph[249] is an open-source storage platform that stores its data in a storage cluster based on RADOS. It is highly scalable and, as a distributed system without a single point of failure, very reliable.

> **Tip**
>
> If you want to quickly set up a basic Ceph cluster, check out MicroCeph[250].

Ceph provides different components for block storage and for file systems.

Ceph Object Gateway[251] is an object storage interface built on top of `librados`[252] to provide applications with a RESTful gateway to Ceph Storage Clusters[253]. It provides object storage functionality with an interface that is compatible with a large subset of the Amazon S3 RESTful API.

Terminology

Ceph uses the term *object* for the data that it stores. The daemon that is responsible for storing and managing data is the *Ceph* OSD. Ceph's storage is divided into *pools*, which are logical partitions for storing objects. They are also referred to as *data pools*, *storage pools* or *OSD pools*.

A *Ceph Object Gateway* consists of several OSD pools and one or more *Ceph Object Gateway daemon* (`radosgw`) processes that provide object gateway functionality.

`cephobject` driver in LXD

> **Note**
>
> The `cephobject` driver can only be used for buckets.
>
> For storage volumes, use the *Ceph* (page 534) or *CephFS* (page 526) drivers.

Unlike other storage drivers, this driver does not set up the storage system but assumes that you already have a Ceph cluster installed.

You must set up a `radosgw` environment beforehand and ensure that its HTTP/HTTPS endpoint URL is reachable from the LXD server or servers. See Manual Deployment[254] for information on how to set up a Ceph cluster and Ceph Object Gateway[255] on how to set up a `radosgw` environment.

The `radosgw` URL can be specified at pool creation time using the *`cephobject.radosgw.endpoint`* (page 533) option.

[249] https://ceph.io/en/
[250] https://canonical.com/microcloud
[251] https://docs.ceph.com/en/latest/radosgw/
[252] https://docs.ceph.com/en/latest/rados/api/librados-intro/
[253] https://docs.ceph.com/en/latest/rados/
[254] https://docs.ceph.com/en/latest/install/manual-deployment/
[255] https://docs.ceph.com/en/latest/radosgw/

LXD uses the `radosgw-admin` command to manage buckets. So this command must be available and operational on the LXD servers.

This driver also behaves differently than other drivers in that it provides remote storage. As a result and depending on the internal network, storage access might be a bit slower than for local storage. On the other hand, using remote storage has big advantages in a cluster setup, because all cluster members have access to the same storage pools with the exact same contents, without the need to synchronize storage pools.

LXD assumes that it has full control over the OSD storage pool. Therefore, you should never maintain any file system entities that are not owned by LXD in a LXD OSD storage pool, because LXD might delete them.

Configuration options

The following configuration options are available for storage pools that use the `cephobject` driver and for storage buckets in these pools.

Storage pool configuration

`cephobject.bucket.name_prefix` Prefix to add to bucket names in Ceph (page 533)

Key:	`cephobject.bucket.name_prefix`
Type:	string
Scope:	global

`cephobject.cluster_name` The Ceph cluster to use (page 533)

Key:	`cephobject.cluster_name`
Type:	string
Scope:	global

`cephobject.radosgw.endpoint` URL of the `radosgw` gateway process (page 533)

Key:	`cephobject.radosgw.endpoint`
Type:	string
Scope:	global

`cephobject.radosgw.endpoint_cert_file` TLS client certificate to use for endpoint communication (page 533)

Key:	`cephobject.radosgw.endpoint_cert_file`
Type:	string
Scope:	global

Specify the path to the file that contains the TLS client certificate.

`cephobject.user.name` The Ceph user to use (page 534)

Key:	`cephobject.user.name`
Type:	string
Default:	`admin`
Scope:	global

`volatile.pool.pristine` Whether the `radosgw lxd-admin` user existed at creation time (page 534)

Key:	`volatile.pool.pristine`
Type:	string
Default:	`true`
Scope:	global

Storage bucket configuration

`size` Quota of the storage bucket (page 534)

Key:	`size`
Type:	string
Scope:	local

Ceph RBD - `ceph`

Ceph[256] is an open-source storage platform that stores its data in a storage cluster based on RADOS. It is highly scalable and, as a distributed system without a single point of failure, very reliable.

Tip

If you want to quickly set up a basic Ceph cluster, check out MicroCeph[257].

Ceph provides different components for block storage and for file systems.

Ceph RBD (RADOS Block Device) is Ceph's block storage component that distributes data and workload across the Ceph cluster. It uses thin provisioning, which means that it is possible to over-commit resources.

[256] https://ceph.io/en/
[257] https://canonical.com/microcloud

Terminology

Ceph uses the term *object* for the data that it stores. The daemon that is responsible for storing and managing data is the *Ceph* OSD. Ceph's storage is divided into *pools*, which are logical partitions for storing objects. They are also referred to as *data pools*, *storage pools* or *OSD pools*.

Ceph block devices are also called *RBD images*, and you can create *snapshots* and *clones* of these RBD images.

ceph driver in LXD

> **Note**
>
> To use the Ceph RBD driver, you must specify it as `ceph`. This is slightly misleading, because it uses only Ceph RBD (block storage) functionality, not full Ceph functionality. For storage volumes with content type `filesystem` (images, containers and custom file-system volumes), the `ceph` driver uses Ceph RBD images with a file system on top (see *`block.filesystem`* (page 538)).
>
> Alternatively, you can use the *CephFS* (page 526) driver to create storage volumes with content type `filesystem`.

Unlike other storage drivers, this driver does not set up the storage system but assumes that you already have a Ceph cluster installed.

This driver also behaves differently than other drivers in that it provides remote storage. As a result and depending on the internal network, storage access might be a bit slower than for local storage. On the other hand, using remote storage has big advantages in a cluster setup, because all cluster members have access to the same storage pools with the exact same contents, without the need to synchronize storage pools.

The `ceph` driver in LXD uses RBD images for images, and snapshots and clones to create instances and snapshots.

LXD assumes that it has full control over the OSD storage pool. Therefore, you should never maintain any file system entities that are not owned by LXD in a LXD OSD storage pool, because LXD might delete them.

Due to the way copy-on-write works in Ceph RBD, parent RBD images can't be removed until all children are gone. As a result, LXD automatically renames any objects that are removed but still referenced. Such objects are kept with a `zombie_` prefix until all references are gone and the object can safely be removed.

Limitations

The `ceph` driver has the following limitations:

Sharing custom volumes between instances

Custom storage volumes with *content type* (page 352) `filesystem` can usually be shared between multiple instances different cluster members. However, because the Ceph RBD driver "simulates" volumes with content type `filesystem` by putting a file system on top of an RBD image, custom storage volumes can only be assigned to a single instance at a time. If you need to share a custom volume with content type `filesystem`, use the *CephFS* (page 526) driver instead.

Sharing the OSD storage pool between installations

Sharing the same OSD storage pool between multiple LXD installations is not supported.

Using an OSD pool of type "erasure"

To use a Ceph OSD pool of type "erasure", you must create the OSD pool beforehand. You must also create a separate OSD pool of type "replicated" that will be used for storing metadata. This is required because Ceph RBD does not support `omap`. To specify which pool is "erasure coded", set the *`ceph.osd.data_pool_name`* (page 536) configuration option to the erasure coded pool name and the *`source`* (page 538) configuration option to the replicated pool name.

Configuration options

The following configuration options are available for storage pools that use the `ceph` driver and for storage volumes in these pools.

Storage pool configuration

`ceph.cluster_name` Name of the Ceph cluster in which to create new storage pools (page 536)

Key:	`ceph.cluster_name`
Type:	string
Default:	`ceph`
Scope:	global

`ceph.osd.data_pool_name` Name of the OSD data pool (page 536)

Key:	`ceph.osd.data_pool_name`
Type:	string
Scope:	global

`ceph.osd.pg_num` Number of placement groups for the OSD storage pool (page 536)

Key:	`ceph.osd.pg_num`
Type:	string
Default:	`32`
Scope:	global

`ceph.osd.pool_name` Name of the OSD storage pool (page 536)

Key:	ceph.osd.pool_name
Type:	string
Default:	name of the pool
Scope:	global

ceph.osd.pool_size Number of RADOS object replicas. Set to 1 for no replication. (page 537)

Key:	ceph.osd.pool_size
Type:	string
Default:	3

This option specifies the name for the file metadata OSD pool that should be used when creating a file system automatically.

ceph.rbd.clone_copy Whether to use RBD lightweight clones (page 537)

Key:	ceph.rbd.clone_copy
Type:	bool
Default:	true
Scope:	global

Enable this option to use RBD lightweight clones rather than full dataset copies.

ceph.rbd.du Whether to use RBD du (page 537)

Key:	ceph.rbd.du
Type:	bool
Default:	true
Scope:	global

This option specifies whether to use RBD du to obtain disk usage data for stopped instances.

ceph.rbd.features Comma-separated list of RBD features to enable on the volumes (page 537)

Key:	ceph.rbd.features
Type:	string
Default:	layering
Scope:	global

ceph.user.name The Ceph user to use when creating storage pools and volumes (page 537)

Key:	`ceph.user.name`
Type:	string
Default:	`admin`
Scope:	global

`source` Existing OSD storage pool to use (page 538)

Key:	`source`
Type:	string
Scope:	local

`volatile.pool.pristine` Whether the pool was empty on creation time (page 538)

Key:	`volatile.pool.pristine`
Type:	string
Default:	`true`
Scope:	global

> **Tip**
>
> In addition to these configurations, you can also set default values for the storage volume configurations. See *Configure default values for storage volumes* (page 190).

Storage volume configuration

`block.filesystem` File system of the storage volume (page 538)

Key:	`block.filesystem`
Type:	string
Default:	same as `volume.block.filesystem`
Condition:	block-based volume with content type `filesystem`
Scope:	global

Valid options are: `btrfs`, `ext4`, `xfs` If not set, `ext4` is assumed.

`block.mount_options` Mount options for block-backed file system volumes (page 538)

Key:	`block.mount_options`
Type:	string
Default:	same as `volume.block.mount_options`
Condition:	block-based volume with content type `filesystem`
Scope:	global

security.shared Enable volume sharing (page 538)

Key:	security.shared
Type:	bool
Default:	same as volume.security.shared or false
Condition:	virtual-machine or custom block volume
Scope:	global

Enabling this option allows sharing the volume across multiple instances despite the possibility of data loss.

security.shifted Enable ID shifting overlay (page 539)

Key:	security.shifted
Type:	bool
Default:	same as volume.security.shifted or false
Condition:	custom volume
Scope:	global

Enabling this option allows attaching the volume to multiple isolated instances.

security.unmapped Disable ID mapping for the volume (page 539)

Key:	security.unmapped
Type:	bool
Default:	same as volume.security.unmappped or false
Condition:	custom volume
Scope:	global

size Size/quota of the storage volume (page 539)

Key:	size
Type:	string
Default:	same as volume.size
Condition:	appropriate driver
Scope:	global

snapshots.expiry When snapshots are to be deleted (page 539)

Key:	snapshots.expiry
Type:	string
Default:	same as volume.snapshots.expiry
Condition:	custom volume
Scope:	global

Specify an expression like `1M 2H 3d 4w 5m 6y`.

snapshots.pattern Template for the snapshot name (page 539)

Key:	`snapshots.pattern`
Type:	string
Default:	same as `volume.snapshots.pattern` or `snap%d`
Condition:	custom volume
Scope:	global

You can specify a naming template that is used for scheduled snapshots and unnamed snapshots.

The `snapshots.pattern` option takes a Pongo2 template string to format the snapshot name.

To add a time stamp to the snapshot name, use the Pongo2 context variable `creation_date`. Make sure to format the date in your template string to avoid forbidden characters in the snapshot name. For example, set `snapshots.pattern` to `{{ creation_date|date:'2006-01-02_15-04-05' }}` to name the snapshots after their time of creation, down to the precision of a second.

Another way to avoid name collisions is to use the placeholder `%d` in the pattern. For the first snapshot, the placeholder is replaced with `0`. For subsequent snapshots, the existing snapshot names are taken into account to find the highest number at the placeholder's position. This number is then incremented by one for the new name.

`snapshots.schedule` Schedule for automatic volume snapshots (page 540)

Key:	`snapshots.schedule`
Type:	string
Default:	same as `snapshots.schedule`
Condition:	custom volume
Scope:	global

Specify either a cron expression (`<minute> <hour> <dom> <month> <dow>`), a comma-separated list of schedule aliases (`@hourly`, `@daily`, `@midnight`, `@weekly`, `@monthly`, `@annually`, `@yearly`), or leave empty to disable automatic snapshots (the default).

`volatile.idmap.last` JSON-serialized UID/GID map that has been applied to the volume (page 540)

Key:	`volatile.idmap.last`
Type:	string
Condition:	filesystem

`volatile.idmap.next` JSON-serialized UID/GID map that has been applied to the volume (page 540)

Key:	`volatile.idmap.next`
Type:	string
Condition:	filesystem

volatile.uuid The volume's UUID (page 540)

Key:	volatile.uuid
Type:	string
Default:	random UUID
Scope:	global

Dell PowerFlex - powerflex

Dell PowerFlex[258] is a software-defined storage solution from Dell Technologies[259]. Among other things it offers the consumption of redundant block storage across the network.

LXD offers access to PowerFlex storage clusters using either NVMe/TCP or Dell's Storage Data Client (SDC). In addition, PowerFlex offers copy-on-write snapshots, thin provisioning and other features.

To use PowerFlex with NVMe/TCP, make sure you have the required kernel modules installed on your host system. On Ubuntu these are `nvme_fabrics` and `nvme_tcp`, which come bundled in the `linux-modules-extra-$(uname -r)` package. LXD takes care of connecting to the respective subsystem.

When using the SDC, LXD requires it to already be connected to the Dell Metadata Manager (MDM) beforehand. As LXD doesn't set up the SDC, follow the official guides from Dell for configuration details.

Terminology

PowerFlex groups various so-called SDS (storage data servers) under logical groups within a protection domain. Those SDS are the hosts that contribute storage capacity to the PowerFlex cluster. A *protection domain* contains storage pools, which represent a set of physical storage devices from different SDS. LXD creates its volumes in those storage pools.

You can take a snapshot of any volume in PowerFlex, which will create an independent copy of the parent volume. PowerFlex volumes get added as a drive to the respective LXD host the volume got mapped to. In case of NVMe/TCP, the LXD host connects to one or multiple NVMe SDT (storage data targets) provided by PowerFlex. Those SDT run as components on the PowerFlex storage layer. In case of SDC, the LXD hosts don't set up any connection by themselves. Instead they depend on the SDC to make the volumes available on the system for consumption.

powerflex driver in LXD

The `powerflex` driver in LXD uses PowerFlex volumes for custom storage volumes, instances and snapshots. For storage volumes with content type `filesystem` (containers and custom file-system volumes), the `powerflex` driver uses volumes with a file system on top (see *block.filesystem* (page 545)). By default, LXD creates thin-provisioned PowerFlex volumes.

LXD expects the PowerFlex protection domain and storage pool already to be set up. Furthermore, LXD assumes that it has full control over the storage pool. Therefore, you should never maintain any volumes that are not owned by LXD in a PowerFlex storage pool, because LXD might delete them.

[258] https://www.dell.com/en-us/shop/powerflex/sf/powerflex
[259] https://www.dell.com/

This driver behaves differently than some of the other drivers in that it provides remote storage. As a result and depending on the internal network, storage access might be a bit slower than for local storage. On the other hand, using remote storage has big advantages in a cluster setup, because all cluster members have access to the same storage pools with the exact same contents, without the need to synchronize storage pools.

When creating a new storage pool using the `powerflex` driver in `nvme` mode, LXD tries to discover one of the SDT from the given storage pool. Alternatively, you can specify which SDT to use with *`powerflex.sdt`* (page 544). LXD instructs the NVMe initiator to connect to all the other SDT when first connecting to the subsystem.

Due to the way copy-on-write works in PowerFlex, snapshots of any volume don't rely on its parent. As a result, volume snapshots are fully functional volumes themselves, and it's possible to take additional snapshots from such volume snapshots. This tree of dependencies is called the *PowerFlex vTree*. Both volumes and their snapshots get added as standalone disks to the LXD host.

Volume names

Due to a *limitation* (page 542) in PowerFlex, volume names cannot exceed 31 characters. Therefore the driver is using the volume's *`volatile.uuid`* (page 548) to generate a fixed length volume name. A UUID of `5a2504b0-6a6c-4849-8ee7-ddb0b674fd14` will render to the base64-encoded string `WiUEsGpsSEm0592wtnT9FA==`.

To be able to identify the volume types and snapshots, special identifiers are prepended to the volume names:

Type	Identifier	Example
Container	`c_`	`c_WiUEsGpsSEm0592wtnT9FA==`
Virtual machine	`v_`	`v_WiUEsGpsSEm0592wtnT9FA==.b`
Image (ISO)	`i_`	`i_WiUEsGpsSEm0592wtnT9FA==.i`
Custom volume	`u_`	`u_WiUEsGpsSEm0592wtnT9FA==`

Limitations

The `powerflex` driver has the following limitations:

Limit of snapshots in a single vTree
: An internal limitation in the PowerFlex vTree does not allow to take more than 126 snapshots of any volume in PowerFlex. This limit also applies to any child of any of the parent volume's snapshots. A single vTree can only have 126 branches.

Non-optimized image storage
: Due to the limit of 126 snapshots in the vTree, the PowerFlex driver doesn't come with support for optimized image storage. This would limit LXD to create only 126 instances from an image. Instead, when launching a new instance, the image's contents get copied to the instance's root volume.

Copying volumes
: PowerFlex does not support creating a copy of the volume so that it gets its own vTree. Therefore, LXD falls back to copying the volume on the local system. This implicates an

increased use of bandwidth due to the volume's contents being transferred over the network twice.

Volume size constraints
In PowerFlex, the size (quota) of a volume must be in multiples of 8 GiB. This results in the smallest possible volume size of 8 GiB. However, if not specified otherwise, volumes are getting thin-provisioned by LXD. PowerFlex volumes can only be increased in size.

Sharing custom volumes between instances
The PowerFlex driver "simulates" volumes with content type `filesystem` by putting a file system on top of a PowerFlex volume. Therefore, custom storage volumes can only be assigned to a single instance at a time.

Sharing the PowerFlex storage pool between installations
Sharing the same PowerFlex storage pool between multiple LXD installations is not supported.

Recovering PowerFlex storage pools
Recovery of PowerFlex storage pools using `lxd recover` is not supported.

Configuration options

The following configuration options are available for storage pools that use the `powerflex` driver and for storage volumes in these pools.

Storage pool configuration

`powerflex.clone_copy` Whether to use non-sparse copies for snapshots (page 543)

Key:	`powerflex.clone_copy`
Type:	bool
Default:	`true`
Scope:	global

If this option is set to `true`, PowerFlex makes a non-sparse copy when creating a snapshot of an instance or custom volume. See *Limitations* (page 542) for more information.

`powerflex.domain` Name of the PowerFlex protection domain (page 543)

Key:	`powerflex.domain`
Type:	string
Scope:	global

This option is required only if *`powerflex.pool`* (page 544) is specified using its name.

`powerflex.gateway` Address of the PowerFlex Gateway (page 543)

Key:	`powerflex.gateway`
Type:	string
Scope:	global

`powerflex.gateway.verify` Whether to verify the PowerFlex Gateway's certificate (page 544)

Key:	`powerflex.gateway.verify`
Type:	bool
Default:	`true`
Scope:	global

`powerflex.mode` How volumes are mapped to the local server (page 544)

Key:	`powerflex.mode`
Type:	string
Default:	the discovered mode
Scope:	global

The mode gets discovered automatically if the system provides the necessary kernel modules. This can be either `nvme` or `sdc`.

`powerflex.pool` ID of the PowerFlex storage pool (page 544)

Key:	`powerflex.pool`
Type:	string
Scope:	global

If you want to specify the storage pool via its name, also set *`powerflex.domain`* (page 543).

`powerflex.sdt` Comma separated list of PowerFlex NVMe/TCP SDTs (page 544)

Key:	`powerflex.sdt`
Type:	string
Scope:	global

`powerflex.user.name` User for PowerFlex Gateway authentication (page 544)

Key:	`powerflex.user.name`
Type:	string
Default:	`admin`
Scope:	global

Must have at least SystemAdmin role to give LXD full control over managed storage pools.

`powerflex.user.password` Password for PowerFlex Gateway authentication (page 545)

Key:	`powerflex.user.password`
Type:	string
Scope:	global

`rsync.bwlimit` Upper limit on the socket I/O for `rsync` (page 545)

Key:	`rsync.bwlimit`
Type:	string
Default:	`0` (no limit)
Scope:	global

When `rsync` must be used to transfer storage entities, this option specifies the upper limit to be placed on the socket I/O.

`rsync.compression` Whether to use compression while migrating storage pools (page 545)

Key:	`rsync.compression`
Type:	bool
Default:	`true`
Scope:	global

`volume.size` Size/quota of the storage volume (page 545)

Key:	`volume.size`
Type:	string
Default:	`8GiB`
Scope:	global

The size must be in multiples of 8 GiB. See *Limitations* (page 542) for more information.

Tip

In addition to these configurations, you can also set default values for the storage volume configurations. See *Configure default values for storage volumes* (page 190).

Storage volume configuration

`block.filesystem` File system of the storage volume (page 545)

Key:	`block.filesystem`
Type:	string
Default:	same as `volume.block.filesystem`
Condition:	block-based volume with content type `filesystem`
Scope:	global

Valid options are: `btrfs`, `ext4`, `xfs` If not set, `ext4` is assumed.

`block.mount_options` Mount options for block-backed file system volumes (page 546)

Key:	`block.mount_options`
Type:	string
Default:	same as `volume.block.mount_options`
Condition:	block-based volume with content type `filesystem`
Scope:	global

`block.type` Whether to create a `thin` or `thick` provisioned volume (page 546)

Key:	`block.type`
Type:	string
Default:	same as `volume.block.type` or `thick`
Scope:	global

`security.shared` Enable volume sharing (page 546)

Key:	`security.shared`
Type:	bool
Default:	same as `volume.security.shared` or `false`
Condition:	virtual-machine or custom block volume
Scope:	global

Enabling this option allows sharing the volume across multiple instances despite the possibility of data loss.

`security.shifted` Enable ID shifting overlay (page 546)

Key:	`security.shifted`
Type:	bool
Default:	same as `volume.security.shifted` or `false`
Condition:	custom volume
Scope:	global

Enabling this option allows attaching the volume to multiple isolated instances.

`security.unmapped` Disable ID mapping for the volume (page 546)

Key:	`security.unmapped`
Type:	bool
Default:	same as `volume.security.unmappped` or `false`
Condition:	custom volume
Scope:	global

`size` Size/quota of the storage volume (page 547)

Key:	`size`
Type:	string
Default:	same as `volume.size`
Scope:	global

The size must be in multiples of 8 GiB. See *Limitations* (page 542) for more information.

`snapshots.expiry` When snapshots are to be deleted (page 547)

Key:	`snapshots.expiry`
Type:	string
Default:	same as `volume.snapshots.expiry`
Condition:	custom volume
Scope:	global

Specify an expression like `1M 2H 3d 4w 5m 6y`.

`snapshots.pattern` Template for the snapshot name (page 547)

Key:	`snapshots.pattern`
Type:	string
Default:	same as `volume.snapshots.pattern` or `snap%d`
Condition:	custom volume
Scope:	global

You can specify a naming template that is used for scheduled snapshots and unnamed snapshots.

The `snapshots.pattern` option takes a Pongo2 template string to format the snapshot name.

To add a time stamp to the snapshot name, use the Pongo2 context variable `creation_date`. Make sure to format the date in your template string to avoid forbidden characters in the snapshot name. For example, set `snapshots.pattern` to `{{ creation_date|date:'2006-01-02_15-04-05' }}` to name the snapshots after their time of creation, down to the precision of a second.

Another way to avoid name collisions is to use the placeholder `%d` in the pattern. For the first snapshot, the placeholder is replaced with `0`. For subsequent snapshots, the existing snapshot names are taken into account to find the highest number at the placeholder's position. This number is then incremented by one for the new name.

`snapshots.schedule` Schedule for automatic volume snapshots (page 547)

Key:	`snapshots.schedule`
Type:	string
Default:	same as `snapshots.schedule`
Condition:	custom volume
Scope:	global

Specify either a cron expression (`<minute> <hour> <dom> <month> <dow>`), a comma-separated list of schedule aliases (`@hourly`, `@daily`, `@midnight`, `@weekly`, `@monthly`, `@annually`, `@yearly`), or leave empty to disable automatic snapshots (the default).

`volatile.idmap.last` JSON-serialized UID/GID map that has been applied to the volume (page 548)

Key:	`volatile.idmap.last`
Type:	string
Condition:	filesystem

`volatile.idmap.next` JSON-serialized UID/GID map that has been applied to the volume (page 548)

Key:	`volatile.idmap.next`
Type:	string
Condition:	filesystem

`volatile.uuid` The volume's UUID (page 548)

Key:	`volatile.uuid`
Type:	string
Default:	random UUID
Scope:	global

Pure Storage - `pure`

Pure Storage[260] is a software-defined storage solution. It offers the consumption of redundant block storage across the network.

LXD supports connecting to Pure Storage storage clusters through two protocols: either iSCSI (Internet Small Computer Systems Interface) or NVMe/TCP (Non-Volatile Memory Express over Transmission Control Protocol). In addition, Pure Storage offers copy-on-write snapshots, thin provisioning, and other features.

To use Pure Storage with LXD requires a Pure Storage API version of at least `2.21`, corresponding to a minimum Purity//FA version of `6.4.2`.

[260] https://www.purestorage.com/

Canonical

Additionally, ensure that the required kernel modules for the selected protocol are installed on your host system. For iSCSI, the iSCSI CLI named `iscsiadm` needs to be installed in addition to the required kernel modules.

Terminology

Each storage pool created in LXD using a Pure Storage driver represents a Pure Storage *pod*, which is an abstraction that groups multiple volumes under a specific name. One benefit of using Pure Storage pods is that they can be linked with multiple Pure Storage arrays to provide additional redundancy.

LXD creates volumes within a pod that is identified by the storage pool name. When the first volume needs to be mapped to a specific LXD host, a corresponding Pure Storage host is created with the name of the LXD host and a suffix of the used protocol. For example, if the LXD host is `host01` and the mode is `nvme`, the resulting Pure Storage host would be `host01-nvme`.

The Pure Storage host is then connected with the required volumes, to allow attaching and accessing volumes from the LXD host. The created Pure Storage host is automatically removed once there are no volumes connected to it.

The `pure` driver in LXD

The `pure` driver in LXD uses Pure Storage volumes for custom storage volumes, instances, and snapshots. All created volumes are thin-provisioned block volumes. If required (for example, for containers and custom file system volumes), LXD formats the volume with a desired file system.

LXD expects Pure Storage to be pre-configured with a specific service (e.g. iSCSI) on network interfaces whose address is provided during storage pool configuration. Furthermore, LXD assumes that it has full control over the Pure Storage pods it manages. Therefore, you should never maintain any volumes in Pure Storage pods that are not owned by LXD because LXD might disconnect or even delete them.

This driver behaves differently than some of the other drivers in that it provides remote storage. As a result, and depending on the internal network, storage access might be a bit slower compared to local storage. On the other hand, using remote storage has significant advantages in a cluster setup: all cluster members have access to the same storage pools with the exact same contents, without the need to synchronize them.

When creating a new storage pool using the `pure` driver in either `iscsi` or `nvme` mode, LXD automatically discovers the array's qualified name and target address (portal). Upon successful discovery, LXD attaches all volumes that are connected to the Pure Storage host that is associated with a specific LXD server. Pure Storage hosts and volume connections are fully managed by LXD.

Volume snapshots are also supported by Pure Storage. However, each snapshot is associated with a parent volume and cannot be directly attached to the host. Therefore, when a snapshot is being exported, LXD creates a temporary volume behind the scenes. This volume is attached to the LXD host and removed once the operation is completed. Similarly, when a volume with at least one snapshot is being copied, LXD sequentially copies snapshots into destination volume, from which a new snapshot is created. Finally, once all snapshots are copied, the source volume is copied into the destination volume.

Volume names

Due to a *limitation* (page 550) in Pure Storage, volume names cannot exceed 63 characters. Therefore, the driver uses the volume's `volatile.uuid` (page 553) to generate a shorter volume name.

For example, a UUID `5a2504b0-6a6c-4849-8ee7-ddb0b674fd14` is first trimmed of any hyphens (`-`), resulting in the string `5a2504b06a6c48498ee7ddb0b674fd14`. To distinguish volume types and snapshots, special identifiers are prepended and appended to the volume names, as depicted in the table below:

Type	Identifier	Example
Container	`c-`	`c-5a2504b06a6c48498ee7ddb0b674fd14`
Virtual machine	`v-`	`v-5a2504b06a6c48498ee7ddb0b674fd14-b` (block volume) and `v-5a2504b06a6c48498ee7ddb0b674fd14` (file system volume)
Image (ISO)	`i-`	`i-5a2504b06a6c48498ee7ddb0b674fd14-i`
Custom volume	`u-`	`u-5a2504b06a6c48498ee7ddb0b674fd14`
Snapshot	`s`	`sc-5a2504b06a6c48498ee7ddb0b674fd14` (container snapshot)

Limitations

The `pure` driver has the following limitations:

Volume size constraints
Minimum volume size (quota) is `1MiB` and must be a multiple of `512B`.

Snapshots cannot be mounted
Snapshots cannot be mounted directly to the host. Instead, a temporary volume must be created to access the snapshot's contents. For internal operations, such as copying instances or exporting snapshots, LXD handles this automatically.

Sharing the Pure Storage storage pool between multiple LXD installations
Sharing a Pure Storage array between multiple LXD installations is possible provided that installations use distinct storage pool names. Storage pools are implemented as Pods on the array and pod names have to be unique.

Recovering Pure Storage storage pools
Recovery of Pure Storage storage pools using `lxd recover` is currently not supported.

Configuration options

The following configuration options are available for storage pools that use the `pure` driver, as well as storage volumes in these pools.

Storage pool configuration

`pure.api.token` API authorization token for Pure Storage gateway (page 551)

Key:	`pure.api.token`
Type:	string

API authorization token for Pure Storage gateway. Must have array_admin role to give LXD full control over managed storage pools (Pure Storage pods).

`pure.gateway` Address of the Pure Storage gateway (page 551)

Key:	`pure.gateway`
Type:	string

`pure.gateway.verify` Whether to verify the Pure Storage gateway's certificate (page 551)

Key:	`pure.gateway.verify`
Type:	bool
Default:	`true`

`pure.mode` How volumes are mapped to the local server (page 551)

Key:	`pure.mode`
Type:	string
Default:	the discovered mode

The mode to use to map Pure Storage volumes to the local server. Supported values are `iscsi` and `nvme`.

`pure.target` List of target addresses. (page 551)

Key:	`pure.target`
Type:	string
Default:	the discovered mode

A comma-separated list of target addresses. If empty, LXD discovers and connects to all available targets. Otherwise, it only connects to the specified addresses.

`volume.size` Size/quota of the storage volume (page 551)

Key:	`volume.size`
Type:	string
Default:	`10GiB`

Default Pure Storage volume size rounded to 512B. The minimum size is 1MiB.

Tip

In addition to these configurations, you can also set default values for the storage volume configurations. See *Configure default values for storage volumes* (page 190).

Storage volume configuration

`block.filesystem` File system of the storage volume (page 552)

Key:	`block.filesystem`
Type:	string
Default:	same as `volume.block.filesystem`
Condition:	block-based volume with content type `filesystem`

Valid options are: `btrfs`, `ext4`, `xfs` If not set, `ext4` is assumed.

`block.mount_options` Mount options for block-backed file system volumes (page 552)

Key:	`block.mount_options`
Type:	string
Default:	same as `volume.block.mount_options`
Condition:	block-based volume with content type `filesystem`

`size` Size/quota of the storage volume (page 552)

Key:	`size`
Type:	string
Default:	same as `volume.size`

Default Pure Storage volume size rounded to 512B. The minimum size is 1MiB.

`snapshots.expiry` When snapshots are to be deleted (page 552)

Key:	`snapshots.expiry`
Type:	string
Default:	same as `volume.snapshots.expiry`
Condition:	custom volume
Scope:	global

Specify an expression like `1M 2H 3d 4w 5m 6y`.

`snapshots.pattern` Template for the snapshot name (page 552)

Key:	`snapshots.pattern`
Type:	string
Default:	same as `volume.snapshots.pattern` or `snap%d`
Condition:	custom volume
Scope:	global

You can specify a naming template that is used for scheduled snapshots and unnamed snapshots.

The `snapshots.pattern` option takes a Pongo2 template string to format the snapshot name.

To add a time stamp to the snapshot name, use the Pongo2 context variable `creation_date`. Make sure to format the date in your template string to avoid forbidden characters in the snapshot name. For example, set `snapshots.pattern` to `{{ creation_date|date:'2006-01-02_15-04-05' }}` to name the snapshots after their time of creation, down to the precision of a second.

Another way to avoid name collisions is to use the placeholder `%d` in the pattern. For the first snapshot, the placeholder is replaced with `0`. For subsequent snapshots, the existing snapshot names are taken into account to find the highest number at the placeholder's position. This number is then incremented by one for the new name.

`snapshots.schedule` Schedule for automatic volume snapshots (page 553)

Key:	`snapshots.schedule`
Type:	string
Default:	same as `snapshots.schedule`
Condition:	custom volume
Scope:	global

Specify either a cron expression (`<minute> <hour> <dom> <month> <dow>`), a comma-separated list of schedule aliases (`@hourly`, `@daily`, `@midnight`, `@weekly`, `@monthly`, `@annually`, `@yearly`), or leave empty to disable automatic snapshots (the default).

`volatile.uuid` The volume's UUID (page 553)

Key:	`volatile.uuid`
Type:	string
Default:	random UUID
Scope:	global

Directory - `dir`

The directory storage driver is a basic backend that stores its data in a standard file and directory structure. This driver is quick to set up and allows inspecting the files directly on the disk, which can be convenient for testing. However, LXD operations are *not optimized* (page 571) for this driver.

`dir` driver in LXD

The `dir` driver in LXD is fully functional and provides the same set of features as other drivers. However, it is much slower than all the other drivers because it must unpack images and do instant copies of instances, snapshots and images.

Unless specified differently during creation (with the `source` configuration option), the data is stored in the `/var/snap/lxd/common/lxd/storage-pools/` (for snap installations) or `/var/lib/lxd/storage-pools/` directory.

Quotas

The `dir` driver supports storage quotas when running on either ext4 or XFS with project quotas enabled at the file system level.

Configuration options

The following configuration options are available for storage pools that use the `dir` driver and for storage volumes in these pools.

Storage pool configuration

`rsync.bwlimit` Upper limit on the socket I/O for `rsync` (page 554)

Key:	`rsync.bwlimit`
Type:	string
Default:	`0` (no limit)
Scope:	global

When `rsync` must be used to transfer storage entities, this option specifies the upper limit to be placed on the socket I/O.

`rsync.compression` Whether to use compression while migrating storage pools (page 554)

Key:	`rsync.compression`
Type:	bool
Default:	`true`
Scope:	global

`source` Path to an existing directory (page 554)

Key:	`source`
Type:	string
Scope:	local

> **Tip**
>
> In addition to these configurations, you can also set default values for the storage volume configurations. See *Configure default values for storage volumes* (page 190).

Storage volume configuration

`security.shared` Enable volume sharing (page 555)

Key:	`security.shared`
Type:	bool
Default:	same as `volume.security.shared` or `false`
Condition:	virtual-machine or custom block volume
Scope:	global

Enabling this option allows sharing the volume across multiple instances despite the possibility of data loss.

`security.shifted` Enable ID shifting overlay (page 555)

Key:	`security.shifted`
Type:	bool
Default:	same as `volume.security.shifted` or `false`
Condition:	custom volume
Scope:	global

Enabling this option allows attaching the volume to multiple isolated instances.

`security.unmapped` Disable ID mapping for the volume (page 555)

Key:	`security.unmapped`
Type:	bool
Default:	same as `volume.security.unmappped` or `false`
Condition:	custom volume
Scope:	global

`size` Size/quota of the storage volume (page 555)

Key:	`size`
Type:	string
Default:	same as `volume.size`
Condition:	appropriate driver
Scope:	global

`snapshots.expiry` When snapshots are to be deleted (page 555)

Key:	`snapshots.expiry`
Type:	string
Default:	same as `volume.snapshots.expiry`
Condition:	custom volume
Scope:	global

Specify an expression like `1M 2H 3d 4w 5m 6y`.

`snapshots.pattern` Template for the snapshot name (page 556)

Key:	`snapshots.pattern`
Type:	string
Default:	same as `volume.snapshots.pattern` or `snap%d`
Condition:	custom volume
Scope:	global

You can specify a naming template that is used for scheduled snapshots and unnamed snapshots.

The `snapshots.pattern` option takes a Pongo2 template string to format the snapshot name.

To add a time stamp to the snapshot name, use the Pongo2 context variable `creation_date`. Make sure to format the date in your template string to avoid forbidden characters in the snapshot name. For example, set `snapshots.pattern` to `{{ creation_date|date:'2006-01-02_15-04-05' }}` to name the snapshots after their time of creation, down to the precision of a second.

Another way to avoid name collisions is to use the placeholder `%d` in the pattern. For the first snapshot, the placeholder is replaced with `0`. For subsequent snapshots, the existing snapshot names are taken into account to find the highest number at the placeholder's position. This number is then incremented by one for the new name.

`snapshots.schedule` Schedule for automatic volume snapshots (page 556)

Key:	`snapshots.schedule`
Type:	string
Default:	same as `snapshots.schedule`
Condition:	custom volume
Scope:	global

Specify either a cron expression (`<minute> <hour> <dom> <month> <dow>`), a comma-separated list of schedule aliases (`@hourly`, `@daily`, `@midnight`, `@weekly`, `@monthly`, `@annually`, `@yearly`), or leave empty to disable automatic snapshots (the default).

`volatile.idmap.last` JSON-serialized UID/GID map that has been applied to the volume (page 556)

Key:	`volatile.idmap.last`
Type:	string
Condition:	filesystem

`volatile.idmap.next` JSON-serialized UID/GID map that has been applied to the volume (page 557)

Key:	`volatile.idmap.next`
Type:	string
Condition:	filesystem

`volatile.uuid` The volume's UUID (page 557)

Key:	`volatile.uuid`
Type:	string
Default:	random UUID
Scope:	global

Storage bucket configuration

To enable storage buckets for local storage pool drivers and allow applications to access the buckets via the S3 protocol, you must configure the *core.storage_buckets_address* (page 404) server setting.

Storage buckets do not have any configuration for `dir` pools. Unlike the other storage pool drivers, the `dir` driver does not support bucket quotas via the `size` setting.

LVM - `lvm`

LVM (Logical Volume Manager) is a storage management framework rather than a file system. It is used to manage physical storage devices, allowing you to create a number of logical storage volumes that use and virtualize the underlying physical storage devices.

Note that it is possible to over-commit the physical storage in the process, to allow flexibility for scenarios where not all available storage is in use at the same time.

To use LVM, make sure you have `lvm2` installed on your machine.

Terminology

LVM can combine several physical storage devices into a *volume group*. You can then allocate *logical volumes* of different types from this volume group.

One supported volume type is a *thin pool*, which allows over-committing the resources by creating thinly provisioned volumes whose total allowed maximum size (quota) is larger than the available physical storage. Another type is a *volume snapshot*, which captures a specific state of a logical volume.

`lvm` driver in LXD

The `lvm` driver in LXD uses logical volumes for images, and volume snapshots for instances and snapshots.

LXD assumes that it has full control over the volume group. Therefore, you should not maintain any file system entities that are not owned by LXD in an LVM volume group, because

LXD might delete them. However, if you need to reuse an existing volume group (for example, because your setup has only one volume group), you can do so by setting the *lvm.vg.force_reuse* (page 559) configuration.

By default, LVM storage pools use an LVM thin pool and create logical volumes for all LXD storage entities (images, instances and custom volumes) in there. This behavior can be changed by setting *lvm.use_thinpool* (page 558) to false when you create the pool. In this case, LXD uses "normal" logical volumes for all storage entities that are not snapshots. Note that this entails serious performance and space reductions for the lvm driver (close to the dir driver both in speed and storage usage). The reason for this is that most storage operations must fall back to using rsync, because logical volumes that are not thin pools do not support snapshots of snapshots. In addition, non-thin snapshots take up much more storage space than thin snapshots, because they must reserve space for their maximum size (quota) at creation time. Therefore, this option should only be chosen if the use case requires it.

For environments with a high instance turnover (for example, continuous integration) you should tweak the backup retain_min and retain_days settings in /etc/lvm/lvm.conf to avoid slowdowns when interacting with LXD.

Configuration options

The following configuration options are available for storage pools that use the lvm driver and for storage volumes in these pools.

Storage pool configuration

lvm.thinpool_metadata_size The size of the thin pool metadata volume (page 558)

Key:	lvm.thinpool_metadata_size
Type:	string
Default:	0 (auto)
Scope:	global

By default, LVM calculates an appropriate size.

lvm.thinpool_name Thin pool where volumes are created (page 558)

Key:	lvm.thinpool_name
Type:	string
Default:	LXDThinPool
Scope:	local

lvm.use_thinpool Whether the storage pool uses a thin pool for logical volumes (page 558)

Key:	lvm.use_thinpool
Type:	bool
Default:	true
Scope:	global

`lvm.vg.force_reuse` Force using an existing non-empty volume group (page 559)

Key:	`lvm.vg.force_reuse`
Type:	bool
Default:	`false`
Scope:	global

`lvm.vg_name` Name of the volume group to create (page 559)

Key:	`lvm.vg_name`
Type:	string
Default:	name of the pool
Scope:	local

`rsync.bwlimit` Upper limit on the socket I/O for `rsync` (page 559)

Key:	`rsync.bwlimit`
Type:	string
Default:	`0` (no limit)
Scope:	global

When `rsync` must be used to transfer storage entities, this option specifies the upper limit to be placed on the socket I/O.

`rsync.compression` Whether to use compression while migrating storage pools (page 559)

Key:	`rsync.compression`
Type:	bool
Default:	`true`
Scope:	global

`size` Size of the storage pool (for loop-based pools) (page 559)

Key:	`size`
Type:	string
Default:	auto (20% of free disk space, >= 5 GiB and <= 30 GiB)
Scope:	local

When creating loop-based pools, specify the size in bytes (*suffixes* (page 506) are supported). You can increase the size to grow the storage pool.

The default (`auto`) creates a storage pool that uses 20% of the free disk space, with a minimum of 5 GiB and a maximum of 30 GiB.

`source` Path to an existing block device, loop file, or LVM volume group (page 559)

Key:	`source`
Type:	string
Scope:	local

`source.wipe` Whether to wipe the block device before creating the pool (page 560)

Key:	`source.wipe`
Type:	bool
Default:	`false`
Scope:	local

Set this option to `true` to wipe the block device specified in `source` prior to creating the storage pool.

Tip

In addition to these configurations, you can also set default values for the storage volume configurations. See *Configure default values for storage volumes* (page 190).

Storage volume configuration

`block.filesystem` File system of the storage volume (page 560)

Key:	`block.filesystem`
Type:	string
Default:	same as `volume.block.filesystem`
Condition:	block-based volume with content type `filesystem`
Scope:	global

Valid options are: `btrfs`, `ext4`, `xfs` If not set, `ext4` is assumed.

`block.mount_options` Mount options for block-backed file system volumes (page 560)

Key:	`block.mount_options`
Type:	string
Default:	same as `volume.block.mount_options`
Condition:	block-based volume with content type `filesystem`
Scope:	global

`lvm.stripes` Number of stripes to use for new volumes (or thin pool volume) (page 560)

Key:	`lvm.stripes`
Type:	string
Default:	same as `volume.lvm.stripes`
Scope:	global

`lvm.stripes.size` Size of stripes to use (page 560)

Key:	`lvm.stripes.size`
Type:	string
Default:	same as `volume.lvm.stripes.size`
Scope:	global

The size must be at least 4096 bytes, and a multiple of 512 bytes.

`security.shared` Enable volume sharing (page 561)

Key:	`security.shared`
Type:	bool
Default:	same as `volume.security.shared` or `false`
Condition:	virtual-machine or custom block volume
Scope:	global

Enabling this option allows sharing the volume across multiple instances despite the possibility of data loss.

`security.shifted` Enable ID shifting overlay (page 561)

Key:	`security.shifted`
Type:	bool
Default:	same as `volume.security.shifted` or `false`
Condition:	custom volume
Scope:	global

Enabling this option allows attaching the volume to multiple isolated instances.

`security.unmapped` Disable ID mapping for the volume (page 561)

Key:	`security.unmapped`
Type:	bool
Default:	same as `volume.security.unmappped` or `false`
Condition:	custom volume
Scope:	global

`size` Size/quota of the storage volume (page 561)

Key:	`size`
Type:	string
Default:	same as `volume.size`
Condition:	appropriate driver
Scope:	global

`snapshots.expiry` When snapshots are to be deleted (page 561)

Key:	`snapshots.expiry`
Type:	string
Default:	same as `volume.snapshots.expiry`
Condition:	custom volume
Scope:	global

Specify an expression like `1M 2H 3d 4w 5m 6y`.

`snapshots.pattern` Template for the snapshot name (page 562)

Key:	`snapshots.pattern`
Type:	string
Default:	same as `volume.snapshots.pattern` or `snap%d`
Condition:	custom volume
Scope:	global

You can specify a naming template that is used for scheduled snapshots and unnamed snapshots.

The `snapshots.pattern` option takes a Pongo2 template string to format the snapshot name.

To add a time stamp to the snapshot name, use the Pongo2 context variable `creation_date`. Make sure to format the date in your template string to avoid forbidden characters in the snapshot name. For example, set `snapshots.pattern` to `{{ creation_date|date:'2006-01-02_15-04-05' }}` to name the snapshots after their time of creation, down to the precision of a second.

Another way to avoid name collisions is to use the placeholder `%d` in the pattern. For the first snapshot, the placeholder is replaced with `0`. For subsequent snapshots, the existing snapshot names are taken into account to find the highest number at the placeholder's position. This number is then incremented by one for the new name.

`snapshots.schedule` Schedule for automatic volume snapshots (page 562)

Key:	`snapshots.schedule`
Type:	string
Default:	same as `snapshots.schedule`
Condition:	custom volume
Scope:	global

Specify either a cron expression (`<minute> <hour> <dom> <month> <dow>`), a comma-separated list of schedule aliases (`@hourly`, `@daily`, `@midnight`, `@weekly`, `@monthly`, `@annually`, `@yearly`), or leave empty to disable automatic snapshots (the default).

`volatile.idmap.last` JSON-serialized UID/GID map that has been applied to the volume (page 562)

Key:	`volatile.idmap.last`
Type:	string
Condition:	filesystem

`volatile.idmap.next` JSON-serialized UID/GID map that has been applied to the volume (page 563)

Key:	`volatile.idmap.next`
Type:	string
Condition:	filesystem

`volatile.uuid` The volume's UUID (page 563)

Key:	`volatile.uuid`
Type:	string
Default:	random UUID
Scope:	global

Storage bucket configuration

To enable storage buckets for local storage pool drivers and allow applications to access the buckets via the S3 protocol, you must configure the *core.storage_buckets_address* (page 404) server setting. `size` Size/quota of the storage bucket (page 563)

Key:	`size`
Type:	string
Default:	same as `volume.size`
Condition:	appropriate driver
Scope:	local

ZFS - `zfs`

ZFS (Zettabyte file system) combines both physical volume management and a file system. A ZFS installation can span across a series of storage devices and is very scalable, allowing you to add disks to expand the available space in the storage pool immediately.

ZFS is a block-based file system that protects against data corruption by using checksums to verify, confirm and correct every operation. To run at a sufficient speed, this mechanism requires a powerful environment with a lot of RAM.

In addition, ZFS offers snapshots and replication, RAID management, copy-on-write clones, compression and other features.

To use ZFS, make sure you have `zfsutils-linux` installed on your machine.

Terminology

ZFS creates logical units based on physical storage devices. These logical units are called *ZFS pools* or *zpools*. Each zpool is then divided into a number of . These can be of different types:

- A can be seen as a partition or a mounted file system.
- A *ZFS volume* represents a block device.

- A *ZFS snapshot* captures a specific state of either a or a ZFS volume. ZFS snapshots are read-only.
- A *ZFS clone* is a writable copy of a ZFS snapshot.

zfs driver in LXD

The zfs driver in LXD uses and ZFS volumes for images and custom storage volumes, and ZFS snapshots and clones to create instances from images and for instance and custom volume snapshots. By default, LXD enables compression when creating a ZFS pool.

LXD assumes that it has full control over the ZFS pool and . Therefore, you should never maintain any or file system entities that are not owned by LXD in a ZFS pool or , because LXD might delete them.

Due to the way copy-on-write works in ZFS, parent can't be removed until all children are gone. As a result, LXD automatically renames any objects that are removed but still referenced. Such objects are kept at a random `deleted/` path until all references are gone and the object can safely be removed. Note that this method might have ramifications for restoring snapshots. See *Limitations* (page 564) below.

LXD automatically enables trimming support on all newly created pools on ZFS 0.8 or later. This increases the lifetime of SSDs by allowing better block re-use by the controller, and it also allows to free space on the root file system when using a loop-backed ZFS pool. If you are running a ZFS version earlier than 0.8 and want to enable trimming, upgrade to at least version 0.8. Then use the following commands to make sure that trimming is automatically enabled for the ZFS pool in the future and trim all currently unused space:

```
zpool upgrade ZPOOL-NAME
zpool set autotrim=on ZPOOL-NAME
zpool trim ZPOOL-NAME
```

Limitations

The `zfs` driver has the following limitations:

Restoring from older snapshots

ZFS doesn't support restoring from snapshots other than the latest one. You can, however, create new instances from older snapshots. This method makes it possible to confirm whether a specific snapshot contains what you need. After determining the correct snapshot, you can *remove the newer snapshots* (page 201) so that the snapshot you need is the latest one and you can restore it.

Alternatively, you can configure LXD to automatically discard the newer snapshots during restore. To do so, set the *`zfs.remove_snapshots`* (page 570) configuration for the volume (or the corresponding `volume.zfs.remove_snapshots` configuration on the storage pool for all volumes in the pool).

Note, however, that if *`zfs.clone_copy`* (page 566) is set to `true`, instance copies use ZFS snapshots too. In that case, you cannot restore an instance to a snapshot taken before the last copy without having to also delete all its descendants. If this is not an option, you can copy the wanted snapshot into a new instance and then delete the old instance. You will, however, lose any other snapshots the instance might have had.

Observing I/O quotas

I/O quotas are unlikely to affect very much. That's because ZFS is a port of a Solaris module (using SPL) and not a native Linux file system using the Linux VFS API, which is where I/O limits are applied.

Feature support in ZFS

Some features, like the use of idmaps or delegation of a ZFS dataset, require ZFS 2.2 or higher and are therefore not widely available yet.

Quotas

ZFS provides two different quota properties: `quota` and `refquota`. `quota` restricts the total size of a , including its snapshots and clones. `refquota` restricts only the size of the data in the , not its snapshots and clones.

By default, LXD uses the `quota` property when you set up a size/quota for your storage volume. If you want to use the `refquota` property instead, set the *`zfs.use_refquota`* (page 570) configuration for the volume (or the corresponding `volume.zfs.use_refquota` configuration on the storage pool for all volumes in the pool).

You can also set the *`zfs.reserve_space`* (page 570) (or `volume.zfs.reserve_space`) configuration to use ZFS `reservation` or `refreservation` along with `quota` or `refquota`.

Configuration options

The following configuration options are available for storage pools that use the `zfs` driver and for storage volumes in these pools.

Storage pool configuration

`size` Size of the storage pool (for loop-based pools) (page 565)

Key:	`size`
Type:	string
Default:	auto (20% of free disk space, >= 5 GiB and <= 30 GiB)
Scope:	local

When creating loop-based pools, specify the size in bytes (*suffixes* (page 506) are supported). You can increase the size to grow the storage pool.

The default (`auto`) creates a storage pool that uses 20% of the free disk space, with a minimum of 5 GiB and a maximum of 30 GiB.

`source` Path to an existing block device, loop file, or ZFS dataset/pool (page 565)

Key:	`source`
Type:	string
Scope:	local

`source.wipe` Whether to wipe the block device before creating the pool (page 565)

Key:	`source.wipe`
Type:	bool
Default:	`false`
Scope:	local

Set this option to `true` to wipe the block device specified in `source` prior to creating the storage pool.

`zfs.clone_copy` Whether to use ZFS lightweight clones (page 566)

Key:	`zfs.clone_copy`
Type:	string
Default:	`true`
Scope:	global

Set this option to `true` or `false` to enable or disable using ZFS lightweight clones rather than full dataset copies. Set the option to `rebase` to copy based on the initial image.

`zfs.export` Whether to export the zpool when an unmount is being performed (page 566)

Key:	`zfs.export`
Type:	bool
Default:	`true`
Scope:	global

`zfs.pool_name` Name of the zpool (page 566)

Key:	`zfs.pool_name`
Type:	string
Default:	name of the pool
Scope:	local

> **Tip**
>
> In addition to these configurations, you can also set default values for the storage volume configurations. See *Configure default values for storage volumes* (page 190).

Storage volume configuration

`block.filesystem` File system of the storage volume (page 566)

Key:	block.filesystem
Type:	string
Default:	same as volume.block.filesystem
Condition:	block-based volume with content type filesystem (zfs.block_mode enabled)
Scope:	global

Valid options are: btrfs, ext4, xfs If not set, ext4 is assumed.

block.mount_options Mount options for block-backed file system volumes (page 567)

Key:	block.mount_options
Type:	string
Default:	same as volume.block.mount_options
Condition:	block-based volume with content type filesystem (zfs.block_mode enabled)
Scope:	global

security.shared Enable volume sharing (page 567)

Key:	security.shared
Type:	bool
Default:	same as volume.security.shared or false
Condition:	virtual-machine or custom block volume
Scope:	global

Enabling this option allows sharing the volume across multiple instances despite the possibility of data loss.

security.shifted Enable ID shifting overlay (page 567)

Key:	security.shifted
Type:	bool
Default:	same as volume.security.shifted or false
Condition:	custom volume
Scope:	global

Enabling this option allows attaching the volume to multiple isolated instances.

security.unmapped Disable ID mapping for the volume (page 567)

Key:	security.unmapped
Type:	bool
Default:	same as volume.security.unmappped or false
Condition:	custom volume
Scope:	global

size Size/quota of the storage volume (page 567)

Key:	`size`
Type:	string
Default:	same as `volume.size`
Condition:	appropriate driver
Scope:	global

`snapshots.expiry` When snapshots are to be deleted (page 568)

Key:	`snapshots.expiry`
Type:	string
Default:	same as `volume.snapshots.expiry`
Condition:	custom volume
Scope:	global

Specify an expression like `1M 2H 3d 4w 5m 6y`.

`snapshots.pattern` Template for the snapshot name (page 568)

Key:	`snapshots.pattern`
Type:	string
Default:	same as `volume.snapshots.pattern` or `snap%d`
Condition:	custom volume
Scope:	global

You can specify a naming template that is used for scheduled snapshots and unnamed snapshots.

The `snapshots.pattern` option takes a Pongo2 template string to format the snapshot name.

To add a time stamp to the snapshot name, use the Pongo2 context variable `creation_date`. Make sure to format the date in your template string to avoid forbidden characters in the snapshot name. For example, set `snapshots.pattern` to `{{ creation_date|date:'2006-01-02_15-04-05' }}` to name the snapshots after their time of creation, down to the precision of a second.

Another way to avoid name collisions is to use the placeholder `%d` in the pattern. For the first snapshot, the placeholder is replaced with `0`. For subsequent snapshots, the existing snapshot names are taken into account to find the highest number at the placeholder's position. This number is then incremented by one for the new name.

`snapshots.schedule` Schedule for automatic volume snapshots (page 568)

Key:	`snapshots.schedule`
Type:	string
Default:	same as `snapshots.schedule`
Condition:	custom volume
Scope:	global

Specify either a cron expression (`<minute> <hour> <dom> <month> <dow>`), a comma-separated list of schedule aliases (`@hourly`, `@daily`, `@midnight`, `@weekly`, `@monthly`, `@annually`, `@yearly`), or leave empty to disable automatic snapshots (the default).

volatile.idmap.last JSON-serialized UID/GID map that has been applied to the volume (page 569)

Key:	volatile.idmap.last
Type:	string
Condition:	filesystem

volatile.idmap.next JSON-serialized UID/GID map that has been applied to the volume (page 569)

Key:	volatile.idmap.next
Type:	string
Condition:	filesystem

volatile.uuid The volume's UUID (page 569)

Key:	volatile.uuid
Type:	string
Default:	random UUID
Scope:	global

zfs.block_mode Whether to use a formatted zvol rather than a dataset (page 569)

Key:	zfs.block_mode
Type:	bool
Default:	same as volume.zfs.block_mode
Scope:	global

zfs.block_mode can be set only for custom storage volumes. To enable ZFS block mode for all storage volumes in the pool, including instance volumes, use volume.zfs.block_mode.

zfs.blocksize Size of the ZFS block (page 569)

Key:	zfs.blocksize
Type:	string
Default:	same as volume.zfs.blocksize
Scope:	global

The size must be between 512 bytes and 16 MiB and must be a power of 2. For a block volume, a maximum value of 128 KiB will be used even if a higher value is set.

Depending on the value of *zfs.block_mode* (page 569), the specified size is used to set either volblocksize or recordsize in ZFS.

zfs.delegate Whether to delegate the ZFS dataset (page 569)

Key:	`zfs.delegate`
Type:	bool
Default:	same as `volume.zfs.delegate`
Condition:	ZFS 2.2 or higher
Scope:	global

This option controls whether to delegate the ZFS dataset and anything underneath it to the container or containers that use it. When used in conjunction with *security.nesting* (page 436), this allows using the `zfs` command in the container.

`zfs.remove_snapshots` Remove snapshots as needed (page 570)

Key:	`zfs.remove_snapshots`
Type:	bool
Default:	same as `volume.zfs.remove_snapshots` or `false`
Scope:	global

`zfs.reserve_space` Use `reservation/refreservation` along with `quota/refquota` (page 570)

Key:	`zfs.reserve_space`
Type:	bool
Default:	same as `volume.zfs.reserve_space` or `false`
Scope:	global

`zfs.use_refquota` Use `refquota` instead of `quota` for space (page 570)

Key:	`zfs.use_refquota`
Type:	bool
Default:	same as `volume.zfs.use_refquota` or `false`
Scope:	global

Storage bucket configuration

To enable storage buckets for local storage pool drivers and allow applications to access the buckets via the S3 protocol, you must configure the *core.storage_buckets_address* (page 404) server setting. `size` Size/quota of the storage bucket (page 570)

Key:	`size`
Type:	string
Default:	same as `volume.size`
Condition:	appropriate driver
Scope:	local

See the corresponding pages for driver-specific information and configuration options.

Feature comparison

Where possible, LXD uses the advanced features of each storage system to optimize operations.

Feature	Directory	Btrfs	LVM	ZFS	Ceph RBD	Cephl	Ceph Object	Dell PowerFlex	Pure Storage
Optimized image storage (page 571)	□	□	□	□	□	□	□	□	□
Optimized instance creation	□	□	□	□	□	□	□	□	□
Optimized snapshot creation	□	□	□	□	□	□	□	□	□
Optimized image transfer	□	□	□	□	□	□	□	□	□
Optimized backup (import/export)	□	□	□	□	□	□	□	□	□
Optimized volume transfer (page 572)	□	□	□	□	□[1]	□	□	□	□
Optimized volume refresh (page 572)	□	□	□[2]	□	□[3]	□	□	□	□
Copy on write	□	□	□	□	□	□	□	□	□
Block based	□	□	□	□	□	□	□	□	□
Instant cloning	□	□	□	□	□	□	□	□	□
Storage driver usable inside a container	□	□	□	□[4]	□	□	□	□	□
Restore from older snapshots (not latest)	□	□	□	□	□	□	□	□	□
Storage quotas	□[5]	□	□	□	□	□	□	□	□
Available on `lxd init`	□	□	□	□	□	□	□	□	□
Object storage	□	□	□	□	□	□	□	□	□

Optimized image storage

Most of the storage drivers have some kind of optimized image storage format. To make instance creation near instantaneous, LXD clones a pre-made image volume when creating an instance rather than unpacking the image tarball from scratch.

To prevent preparing such a volume on a storage pool that might never be used with that image, the volume is generated on demand. Therefore, the first instance takes longer to create than subsequent ones.

[1] Volumes of type `block` will fall back to non-optimized transfer when migrating to an older LXD server that doesn't yet support the `RBD_AND_RSYNC` migration type.

[2] Requires *`lvm.use_thinpool`* (page 558) to be enabled. Only when refreshing local volumes.

[3] Only for volumes of type `block`.

[4] Requires *`zfs.delegate`* (page 569) to be enabled.

[5] The `dir` driver supports storage quotas when running on either ext4 or XFS with project quotas enabled at the file system level.

Optimized volume transfer

Btrfs, ZFS and Ceph RBD have an internal send/receive mechanism that allows for optimized volume transfer.

LXD uses this optimized transfer when transferring instances and snapshots between storage pools that use the same storage driver, if the storage driver supports optimized transfer and the optimized transfer is actually quicker. Otherwise, LXD uses `rsync` to transfer container and file system volumes, or raw block transfer to transfer virtual machine and custom block volumes.

The optimized transfer uses the underlying storage driver's native functionality for transferring data, which is usually faster than using `rsync` or raw block transfer.

Optimized volume refresh

The full potential of the optimized transfer becomes apparent when refreshing a copy of an instance or custom volume that uses periodic snapshots. If the optimized transfer isn't supported by the driver or its implementation of volume refresh, instead of the delta, the entire volume including its snapshot(s) will be copied using either `rsync` or raw block transfer. LXD will try to keep the overhead low by transferring only the volume itself or any snapshots that are missing on the target.

When optimized refresh is available for an instance or custom volume, LXD bases the refresh on the latest snapshot, which means:

- When you take a first snapshot and refresh the copy, the transfer will take roughly the same time as a full copy. LXD transfers the new snapshot and the difference between the snapshot and the main volume.
- For subsequent snapshots, the transfer is considerably faster. LXD does not transfer the full new snapshot, but only the difference between the new snapshot and the latest snapshot that already exists on the target.
- When refreshing without a new snapshot, LXD transfers only the differences between the main volume and the latest snapshot on the target. This transfer is usually faster than using `rsync` (as long as the latest snapshot is not too outdated).

On the other hand, refreshing copies of instances without snapshots (either because the instance doesn't have any snapshots or because the refresh uses the `--instance-only` flag) would actually be slower than using `rsync` or raw block transfer. In such cases, the optimized transfer would transfer the difference between the (non-existent) latest snapshot and the main volume, thus the full volume. Therefore, LXD uses `rsync` or raw block transfer instead of the optimized transfer for refreshes without snapshots.

Recommended setup

The two best options for use with LXD are ZFS and Btrfs. They have similar functionalities, but ZFS is more reliable.

Whenever possible, you should dedicate a full disk or partition to your LXD storage pool. LXD allows to create loop-based storage, but this isn't recommended for production use. See *Data storage location* (page 350) for more information.

The directory backend should be considered as a last resort option. It supports all main LXD features, but is slow and inefficient because it cannot perform instant copies or snapshots.

Therefore, it constantly copies the instance's full storage.

Security considerations

Currently, the Linux kernel might silently ignore mount options and not apply them when a block-based file system (for example, `ext4`) is already mounted with different mount options. This means when dedicated disk devices are shared between different storage pools with different mount options set, the second mount might not have the expected mount options. This becomes security relevant when, for example, one storage pool is supposed to provide `acl` support and the second one is supposed to not provide `acl` support.

For this reason, it is currently recommended to either have dedicated disk devices per storage pool or to ensure that all storage pools that share the same dedicated disk device use the same mount options.

Related topics

How-to guides:

- *Storage* (page 175)

Explanation:

- *Storage pools, volumes, and buckets* (page 349)

4.2.7. Networks

LXD supports different network types for *Managed networks* (page 354).

Fully controlled networks

Fully controlled networks create network interfaces and provide most functionality, including, for example, the ability to do IP management.

LXD supports the following network types:

Bridge network

As one of the possible network configuration types under LXD, LXD supports creating and managing network bridges.

A network bridge creates a virtual L2 Ethernet switch that instance NICs can connect to, making it possible for them to communicate with each other and the host. LXD bridges can leverage underlying native Linux bridges and Open vSwitch.

The `bridge` network type allows to create an L2 bridge that connects the instances that use it together into a single network L2 segment. Bridges created by LXD are managed, which means that in addition to creating the bridge interface itself, LXD also sets up a local `dnsmasq` process to provide DHCP, IPv6 route announcements and DNS services to the network. By default, it also performs NAT for the bridge.

See *How to configure your firewall* (page 258) for instructions on how to configure your firewall to work with LXD bridge networks.

Note

Static DHCP assignments depend on the client using its MAC address as the DHCP identifier. This method prevents conflicting leases when copying an instance, and thus makes statically assigned leases work properly.

IPv6 prefix size

If you're using IPv6 for your bridge network, you should use a prefix size of 64.

Larger subnets (i.e., using a prefix smaller than 64) should work properly too, but they aren't typically that useful for SLAAC (Stateless Address Auto-configuration).

Smaller subnets are in theory possible (when using stateful DHCPv6 for IPv6 allocation), but they aren't properly supported by `dnsmasq` and might cause problems. If you must create a smaller subnet, use static allocation or another standalone router advertisement daemon.

Configuration options

The following configuration key namespaces are currently supported for the `bridge` network type:

- `bgp` (BGP peer configuration)
- `bridge` (L2 interface configuration)
- `dns` (DNS server and resolution configuration)
- `fan` (configuration specific to the Ubuntu FAN overlay)
- `ipv4` (L3 IPv4 configuration)
- `ipv6` (L3 IPv6 configuration)
- `maas` (MAAS network identification)
- `security` (network ACL configuration)
- `raw` (raw configuration file content)
- `tunnel` (cross-host tunneling configuration)
- `user` (free-form key/value for user metadata)

Note

LXD uses the CIDR notation[261] where network subnet information is required, for example, `192.0.2.0/24` or `2001:db8::/32`. This does not apply to cases where a single address is required, for example, local/remote addresses of tunnels, NAT addresses or specific addresses to apply to an instance.

The following configuration options are available for the `bridge` network type: `bgp.ipv4.nexthop` Override the IPv4 next-hop for advertised prefixes (page 574)

[261] https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing

Key:	`bgp.ipv4.nexthop`
Type:	string
Default:	local address
Condition:	BGP server
Scope:	local

`bgp.ipv6.nexthop` Override the IPv6 next-hop for advertised prefixes (page 575)

Key:	`bgp.ipv6.nexthop`
Type:	string
Default:	local address
Condition:	BGP server
Scope:	local

`bgp.peers.NAME.address` Peer address (IPv4 or IPv6) (page 575)

Key:	`bgp.peers.NAME.address`
Type:	string
Condition:	BGP server
Scope:	global

`bgp.peers.NAME.asn` Peer AS number (page 575)

Key:	`bgp.peers.NAME.asn`
Type:	integer
Condition:	BGP server
Scope:	global

`bgp.peers.NAME.holdtime` Peer session hold time (page 575)

Key:	`bgp.peers.NAME.holdtime`
Type:	integer
Default:	`180`
Condition:	BGP server
Required:	no
Scope:	global

Specify the hold time in seconds.

`bgp.peers.NAME.password` Peer session password (page 575)

Key:	`bgp.peers.NAME.password`
Type:	string
Default:	(no password)
Condition:	BGP server
Required:	no
Scope:	global

`bridge.driver` Bridge driver (page 576)

Key:	`bridge.driver`
Type:	string
Default:	`native`
Scope:	global

Possible values are `native` and `openvswitch`.

`bridge.external_interfaces` Unconfigured network interfaces to include in the bridge (page 576)

Key:	`bridge.external_interfaces`
Type:	string
Scope:	local

Specify a comma-separated list of unconfigured network interfaces to include in the bridge.

`bridge.hwaddr` MAC address for the bridge (page 576)

Key:	`bridge.hwaddr`
Type:	string
Scope:	global

`bridge.mode` Bridge operation mode (page 576)

Key:	`bridge.mode`
Type:	string
Default:	`standard`
Scope:	global

Possible values are `standard` and `fan`.

`bridge.mtu` Bridge MTU (page 576)

Key:	bridge.mtu
Type:	integer
Default:	1500 if bridge.mode=standard, 1480 if bridge.mode=fan and fan.type=ipip, or 1450 if bridge.mode=fan and fan.type=vxlan
Scope:	global

The default value varies depending on whether the bridge uses a tunnel or a fan setup.

dns.domain Domain to advertise to DHCP clients and use for DNS resolution (page 577)

Key:	dns.domain
Type:	string
Default:	lxd
Scope:	global

dns.mode DNS registration mode (page 577)

Key:	dns.mode
Type:	string
Default:	managed
Scope:	global

Possible values are none for no DNS record, managed for LXD-generated static records, and dynamic for client-generated records.

dns.search Full domain search list (page 577)

Key:	dns.search
Type:	string
Default:	dns.domain value
Scope:	global

Specify a comma-separated list of domains.

dns.zone.forward DNS zone names for forward DNS records (page 577)

Key:	dns.zone.forward
Type:	string
Scope:	global

Specify a comma-separated list of DNS zone names.

dns.zone.reverse.ipv4 DNS zone name for IPv4 reverse DNS records (page 577)

Key:	`dns.zone.reverse.ipv4`
Type:	string
Scope:	global

`dns.zone.reverse.ipv6` DNS zone name for IPv6 reverse DNS records (page 578)

Key:	`dns.zone.reverse.ipv6`
Type:	string
Scope:	global

`fan.overlay_subnet` Subnet to use as the overlay for the FAN (page 578)

Key:	`fan.overlay_subnet`
Type:	string
Default:	`240.0.0.0/8`
Condition:	fan mode
Scope:	global

Use CIDR notation.

`fan.type` Tunneling type for the FAN (page 578)

Key:	`fan.type`
Type:	string
Default:	`vxlan`
Condition:	fan mode
Scope:	global

Possible values are `vxlan` and `ipip`.

`fan.underlay_subnet` Subnet to use as the underlay for the FAN (page 578)

Key:	`fan.underlay_subnet`
Type:	string
Default:	initial value on creation: `auto`
Condition:	fan mode
Scope:	global

Use CIDR notation.

You can set the option to `auto` to use the default gateway subnet.

`ipv4.address` IPv4 address for the bridge (page 578)

Key:	`ipv4.address`
Type:	string
Default:	initial value on creation: `auto`
Condition:	standard mode
Scope:	global

Use CIDR notation.

You can set the option to `none` to turn off IPv4, or to `auto` to generate a new random unused subnet.

`ipv4.dhcp` Whether to allocate IPv4 addresses using DHCP (page 579)

Key:	`ipv4.dhcp`
Type:	bool
Default:	`true`
Condition:	IPv4 address
Scope:	global

`ipv4.dhcp.expiry` When to expire DHCP leases (page 579)

Key:	`ipv4.dhcp.expiry`
Type:	string
Default:	`1h`
Condition:	IPv4 DHCP
Scope:	global

`ipv4.dhcp.gateway` Address of the gateway for the IPv4 subnet (page 579)

Key:	`ipv4.dhcp.gateway`
Type:	string
Default:	IPv4 address
Condition:	IPv4 DHCP
Scope:	global

`ipv4.dhcp.ranges` IPv4 ranges to use for DHCP (page 579)

Key:	`ipv4.dhcp.ranges`
Type:	string
Default:	all addresses
Condition:	IPv4 DHCP
Scope:	global

Specify a comma-separated list of IPv4 ranges in FIRST-LAST format.

`ipv4.firewall` Whether to generate filtering firewall rules for this network (page 579)

Key:	`ipv4.firewall`
Type:	bool
Default:	`true`
Condition:	IPv4 address
Scope:	global

`ipv4.nat` Whether to use NAT for IPv4 (page 580)

Key:	`ipv4.nat`
Type:	bool
Default:	`false` (initial value on creation if `ipv4.address` is set to `auto`: `true`)
Condition:	IPv4 address
Scope:	global

`ipv4.nat.address` Source address used for outbound traffic from the bridge (page 580)

Key:	`ipv4.nat.address`
Type:	string
Condition:	IPv4 address
Scope:	global

`ipv4.nat.order` Where to add the required NAT rules (page 580)

Key:	`ipv4.nat.order`
Type:	string
Default:	`before`
Condition:	IPv4 address
Scope:	global

Set this option to `before` to add the NAT rules before any pre-existing rules, or to `after` to add them after the pre-existing rules.

`ipv4.ovn.ranges` IPv4 ranges to use for child OVN network routers (page 580)

Key:	`ipv4.ovn.ranges`
Type:	string
Scope:	global

Specify a comma-separated list of IPv4 ranges in FIRST-LAST format.

`ipv4.routes` Additional IPv4 CIDR subnets to route to the bridge (page 580)

Key:	`ipv4.routes`
Type:	string
Condition:	IPv4 address
Scope:	global

Specify a comma-separated list of IPv4 CIDR subnets.

`ipv4.routing` Whether to route IPv4 traffic in and out of the bridge (page 581)

Key:	`ipv4.routing`
Type:	bool
Default:	`true`
Condition:	IPv4 address
Scope:	global

`ipv6.address` IPv6 address for the bridge (page 581)

Key:	`ipv6.address`
Type:	string
Default:	initial value on creation: `auto`
Condition:	standard mode
Scope:	global

Use CIDR notation.

You can set the option to `none` to turn off IPv6, or to `auto` to generate a new random unused subnet.

`ipv6.dhcp` Whether to provide additional network configuration over DHCP (page 581)

Key:	`ipv6.dhcp`
Type:	bool
Default:	`true`
Condition:	IPv6 address
Scope:	global

`ipv6.dhcp.expiry` When to expire DHCP leases (page 581)

Key:	`ipv6.dhcp.expiry`
Type:	string
Default:	`1h`
Condition:	IPv6 DHCP
Scope:	global

`ipv6.dhcp.ranges` IPv6 ranges to use for DHCP (page 581)

Key:	`ipv6.dhcp.ranges`
Type:	string
Default:	all addresses
Condition:	IPv6 stateful DHCP
Scope:	global

Specify a comma-separated list of IPv6 ranges in FIRST-LAST format.

`ipv6.dhcp.stateful` Whether to allocate IPv6 addresses using DHCP (page 582)

Key:	`ipv6.dhcp.stateful`
Type:	bool
Default:	`false`
Condition:	IPv6 DHCP
Scope:	global

`ipv6.firewall` Whether to generate filtering firewall rules for this network (page 582)

Key:	`ipv6.firewall`
Type:	bool
Default:	`true`
Condition:	IPv6 DHCP
Scope:	global

`ipv6.nat` Whether to use NAT for IPv6 (page 582)

Key:	`ipv6.nat`
Type:	bool
Default:	`false` (initial value on creation if `ipv6.address` is set to `auto`: `true`)
Condition:	IPv6 address
Scope:	global

`ipv6.nat.address` Source address used for outbound traffic from the bridge (page 582)

Key:	`ipv6.nat.address`
Type:	string
Condition:	IPv6 address
Scope:	global

`ipv6.nat.order` Where to add the required NAT rules (page 582)

Key:	`ipv6.nat.order`
Type:	string
Default:	`before`
Condition:	IPv6 address
Scope:	global

Set this option to `before` to add the NAT rules before any pre-existing rules, or to `after` to add them after the pre-existing rules.

`ipv6.ovn.ranges` IPv6 ranges to use for child OVN network routers (page 583)

Key:	`ipv6.ovn.ranges`
Type:	string
Scope:	global

Specify a comma-separated list of IPv6 ranges in FIRST-LAST format.

`ipv6.routes` Additional IPv6 CIDR subnets to route to the bridge (page 583)

Key:	`ipv6.routes`
Type:	string
Condition:	IPv6 address
Scope:	global

Specify a comma-separated list of IPv6 CIDR subnets.

`ipv6.routing` Whether to route IPv6 traffic in and out of the bridge (page 583)

Key:	`ipv6.routing`
Type:	bool
Condition:	IPv6 address
Scope:	global

`maas.subnet.ipv4` MAAS IPv4 subnet to register instances in (page 583)

Key:	`maas.subnet.ipv4`
Type:	string
Condition:	IPv4 address; using the `network` property on the NIC
Scope:	global

`maas.subnet.ipv6` MAAS IPv6 subnet to register instances in (page 583)

Key:	`maas.subnet.ipv6`
Type:	string
Condition:	IPv6 address; using the `network` property on the NIC
Scope:	global

`raw.dnsmasq` Additional `dnsmasq` configuration to append to the configuration file (page 584)

Key:	`raw.dnsmasq`
Type:	string
Scope:	global

`security.acls` Network ACLs to apply to NICs connected to this network (page 584)

Key:	`security.acls`
Type:	string
Scope:	global

Specify a comma-separated list of network ACLs.

Also see *Bridge limitations* (page 234).

`security.acls.default.egress.action` Default action to use for egress traffic (page 584)

Key:	`security.acls.default.egress.action`
Type:	string
Condition:	`security.acls`
Scope:	global

The specified action is used for all egress traffic that doesn't match any ACL rule.

`security.acls.default.egress.logged` Whether to log egress traffic that doesn't match any ACL rule (page 584)

Key:	`security.acls.default.egress.logged`
Type:	bool
Condition:	`security.acls`
Scope:	global

`security.acls.default.ingress.action` Default action to use for ingress traffic (page 584)

Key:	`security.acls.default.ingress.action`
Type:	string
Condition:	`security.acls`
Scope:	global

The specified action is used for all ingress traffic that doesn't match any ACL rule.

security.acls.default.ingress.logged Whether to log ingress traffic that doesn't match any ACL rule (page 585)

Key:	security.acls.default.ingress.logged
Type:	bool
Condition:	security.acls
Scope:	global

tunnel.NAME.group Multicast address for vxlan (page 585)

Key:	tunnel.NAME.group
Type:	string
Condition:	vxlan

This address is used if *tunnel.NAME.local* (page 585) and *tunnel.NAME.remote* (page 586) aren't set.

tunnel.NAME.id Specific tunnel ID to use for the vxlan tunnel (page 585)

Key:	tunnel.NAME.id
Type:	integer
Condition:	vxlan

tunnel.NAME.interface Specific host interface to use for the tunnel (page 585)

Key:	tunnel.NAME.interface
Type:	string
Condition:	vxlan

tunnel.NAME.local Local address for the tunnel (page 585)

Key:	tunnel.NAME.local
Type:	string
Condition:	gre or vxlan
Required:	not required for multicast vxlan

tunnel.NAME.port Specific port to use for the vxlan tunnel (page 585)

Key:	`tunnel.NAME.port`
Type:	integer
Default:	`0`
Condition:	`vxlan`

`tunnel.NAME.protocol` Tunneling protocol (page 586)

Key:	`tunnel.NAME.protocol`
Type:	string
Condition:	standard mode

Possible values are `vxlan` and `gre`.

`tunnel.NAME.remote` Remote address for the tunnel (page 586)

Key:	`tunnel.NAME.remote`
Type:	string
Condition:	`gre` or `vxlan`
Required:	not required for multicast `vxlan`

`tunnel.NAME.ttl` Specific TTL to use for multicast routing topologies (page 586)

Key:	`tunnel.NAME.ttl`
Type:	string
Default:	`1`
Condition:	`vxlan`

`user.*` User-provided free-form key/value pairs (page 586)

Key:	`user.*`
Type:	string
Scope:	global

Supported features

The following features are supported for the `bridge` network type:

- *How to configure network ACLs* (page 216)
- *How to configure network forwards* (page 235)
- *How to configure network zones* (page 252)
- *How to configure LXD as a BGP server* (page 214)

- *How to integrate with systemd-resolved* (page 262)

Firewall issues

See *How to configure your firewall* (page 258) for instructions on how to troubleshoot firewall issues.

OVN network

OVN is a software-defined networking system that supports virtual network abstraction. You can use it to build your own private cloud. See `www.ovn.org`[262] for more information.

The ovn network type allows to create logical networks using the OVN SDN (software-defined networking). This kind of network can be useful for labs and multi-tenant environments where the same logical subnets are used in multiple discrete networks.

A LXD OVN network can be connected to an existing managed *Bridge network* (page 573) or *Physical network* (page 595) to gain access to the wider network. By default, all connections from the OVN logical networks are NATed to an IP allocated from the uplink network.

See *How to set up OVN with LXD* (page 266) for basic instructions for setting up an OVN network.

> **Note**
>
> Static DHCP assignments depend on the client using its MAC address as the DHCP identifier. This method prevents conflicting leases when copying an instance, and thus makes statically assigned leases work properly.

OVN networking architecture

The following figure shows the OVN network traffic flow in a LXD cluster:

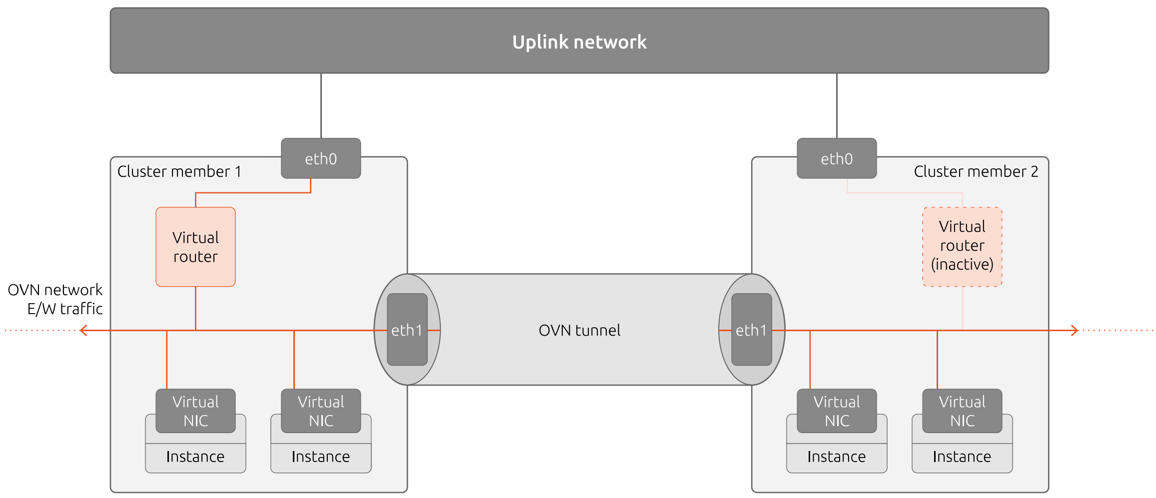


Fig. 1: OVN networking (one network)

[262] https://www.ovn.org/

The OVN network connects the different cluster members. Network traffic between the cluster members passes through the NIC for inter-cluster traffic (`eth1` in the figure) and is transmitted through an OVN tunnel. This traffic between cluster members is referred to as *OVN east/west traffic*.

For outside connectivity, the OVN network requires an uplink network (a *Bridge network* (page 573) or a *Physical network* (page 595)). The OVN network uses a virtual router to connect to the uplink network through the NIC for uplink traffic (`eth0` in the figure). The virtual router is active on only one of the cluster members, and can move to a different member at any time. Independent of where the router resides, the OVN network is available on all cluster members.

Every instance on any cluster member can connect to the OVN network through its virtual NIC (usually `eth0` for containers and `enp5s0` for virtual machines). The traffic between the instances and the uplink network is referred to as *OVN north/south traffic*.

The strengths of using OVN become apparent when looking at a networking architecture with more than one OVN network:

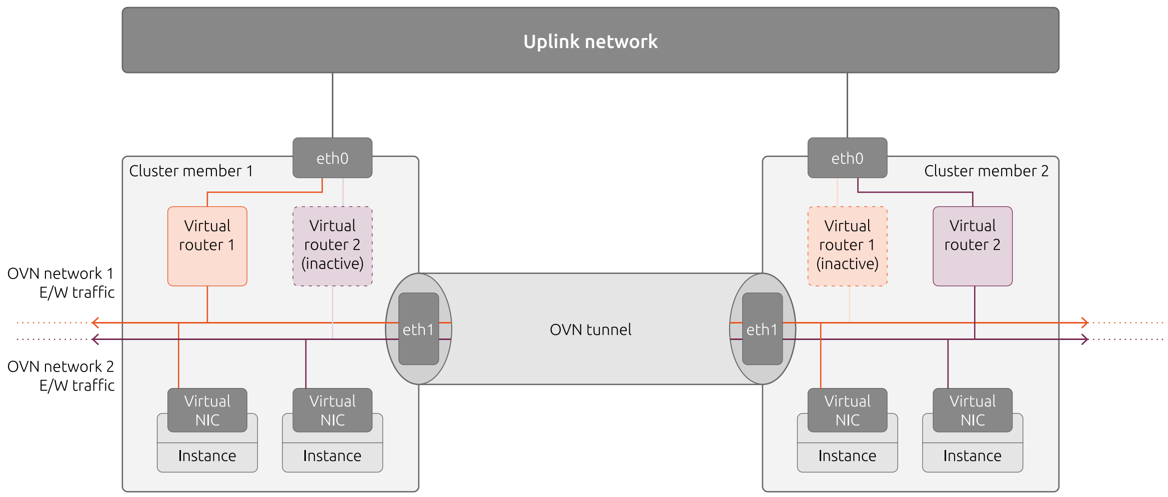


Fig. 2: OVN networking (two networks)

In this case, both depicted OVN networks are completely independent. Both networks are available on all cluster members (with each virtual router being active on one random cluster member). Each instance can use either of the networks, and the traffic on either network is completely isolated from the other network.

Configuration options

The following configuration key namespaces are currently supported for the `ovn` network type:

- `bridge` (L2 interface configuration)
- `dns` (DNS server and resolution configuration)
- `ipv4` (L3 IPv4 configuration)
- `ipv6` (L3 IPv6 configuration)
- `security` (network ACL configuration)

- `user` (free-form key/value for user metadata)

Note

LXD uses the CIDR notation[263] where network subnet information is required, for example, `192.0.2.0/24` or `2001:db8::/32`. This does not apply to cases where a single address is required, for example, local/remote addresses of tunnels, NAT addresses or specific addresses to apply to an instance.

The following configuration options are available for the `ovn` network type: `bridge.hwaddr` MAC address for the bridge (page 589)

Key:	`bridge.hwaddr`
Type:	string

`bridge.mtu` Bridge MTU (page 589)

Key:	`bridge.mtu`
Type:	integer
Default:	`1442`

The default value allows the host to host Geneve tunnels.

`dns.domain` Domain to advertise to DHCP clients and use for DNS resolution (page 589)

Key:	`dns.domain`
Type:	string
Default:	`lxd`

`dns.search` Full domain search list (page 589)

Key:	`dns.search`
Type:	string
Default:	`dns.domain` value

Specify a comma-separated list of domains.

`dns.zone.forward` DNS zone names for forward DNS records (page 589)

Key:	`dns.zone.forward`
Type:	string

Specify a comma-separated list of DNS zone names.

[263] https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing

`dns.zone.reverse.ipv4` DNS zone name for IPv4 reverse DNS records (page 589)

Key:	`dns.zone.reverse.ipv4`
Type:	string

`dns.zone.reverse.ipv6` DNS zone name for IPv6 reverse DNS records (page 590)

Key:	`dns.zone.reverse.ipv6`
Type:	string

`ipv4.address` IPv4 address for the OVN network (page 590)

Key:	`ipv4.address`
Type:	string
Default:	initial value on creation: `auto`
Condition:	standard mode

Use CIDR notation.

You can set the option to `none` to turn off IPv4, or to `auto` to generate a new random unused subnet.

`ipv4.dhcp` Whether to allocate IPv4 addresses using DHCP (page 590)

Key:	`ipv4.dhcp`
Type:	bool
Default:	`true`
Condition:	IPv4 address

`ipv4.l3only` Whether to enable layer 3 only mode for IPv4 (page 590)

Key:	`ipv4.l3only`
Type:	bool
Default:	`false`
Condition:	IPv4 address

`ipv4.nat` Whether to use NAT for IPv4 (page 590)

Key:	`ipv4.nat`
Type:	bool
Default:	`false` (initial value on creation if `ipv4.address` is set to `auto`: `true`)
Condition:	IPv4 address

`ipv4.nat.address` Source address used for outbound traffic from the network (page 590)

Key:	`ipv4.nat.address`
Type:	string
Condition:	IPv4 address; requires uplink `ovn.ingress_mode=routed`

`ipv6.address` IPv6 address for the OVN network (page 591)

Key:	`ipv6.address`
Type:	string
Default:	initial value on creation: `auto`
Condition:	standard mode

Use CIDR notation.

You can set the option to `none` to turn off IPv6, or to `auto` to generate a new random unused subnet.

`ipv6.dhcp` Whether to provide additional network configuration over DHCP (page 591)

Key:	`ipv6.dhcp`
Type:	bool
Default:	`true`
Condition:	IPv6 address

`ipv6.dhcp.stateful` Whether to allocate IPv6 addresses using DHCP (page 591)

Key:	`ipv6.dhcp.stateful`
Type:	bool
Default:	`false`
Condition:	IPv6 DHCP

`ipv6.l3only` Whether to enable layer 3 only mode for IPv6 (page 591)

Key:	`ipv6.l3only`
Type:	bool
Default:	`false`
Condition:	IPv6 DHCP stateful

`ipv6.nat` Whether to use NAT for IPv6 (page 591)

Key:	`ipv6.nat`
Type:	bool
Default:	`false` (initial value on creation if `ipv6.address` is set to `auto`: `true`)
Condition:	IPv6 address

`ipv6.nat.address` Source address used for outbound traffic from the network (page 591)

Key:	`ipv6.nat.address`
Type:	string
Condition:	IPv6 address; requires uplink `ovn.ingress_mode=routed`

`network` Uplink network to use for external network access (page 592)

Key:	`network`
Type:	string

`security.acls` Network ACLs to apply to NICs connected to this network (page 592)

Key:	`security.acls`
Type:	string

Specify a comma-separated list of network ACLs.

`security.acls.default.egress.action` Default action to use for egress traffic (page 592)

Key:	`security.acls.default.egress.action`
Type:	string
Default:	`reject`
Condition:	`security.acls`

The specified action is used for all egress traffic that doesn't match any ACL rule.

`security.acls.default.egress.logged` Whether to log egress traffic that doesn't match any ACL rule (page 592)

Key:	`security.acls.default.egress.logged`
Type:	bool
Default:	`false`
Condition:	`security.acls`

`security.acls.default.ingress.action` Default action to use for ingress traffic (page 592)

Key:	`security.acls.default.ingress.action`
Type:	string
Default:	`reject`
Condition:	`security.acls`

The specified action is used for all ingress traffic that doesn't match any ACL rule.

`security.acls.default.ingress.logged` Whether to log ingress traffic that doesn't match any ACL rule (page 592)

Key:	`security.acls.default.ingress.logged`
Type:	bool
Default:	`false`
Condition:	`security.acls`

`user.*` User-provided free-form key/value pairs (page 593)

Key:	`user.*`
Type:	string

Supported features

The following features are supported for the `ovn` network type:

- *How to configure network ACLs* (page 216)
- *How to configure network forwards* (page 235)
- *How to configure network zones* (page 252)
- *How to create OVN peer routing relationships* (page 276)
- *How to configure network load balancers* (page 271)

External networks

External networks use network interfaces that already exist. Therefore, LXD has limited possibility to control them, and LXD features like network ACLs, network forwards and network zones are not supported.

The main purpose for using external networks is to provide an uplink network through a parent interface. This external network specifies the presets to use when connecting instances or other networks to a parent interface.

LXD supports the following external network types:

Macvlan network

Macvlan is a virtual LAN that you can use if you want to assign several IP addresses to the same network interface, basically splitting up the network interface into several subinterfaces with their own IP addresses. You can then assign IP addresses based on the randomly generated MAC addresses.

The `macvlan` network type allows to specify presets to use when connecting instances to a parent interface. In this case, the instance NICs can simply set the `network` option to the network they connect to without knowing any of the underlying configuration details.

Note

If you are using a `macvlan` network, communication between the LXD host and the instances is not possible. Both the host and the instances can talk to the gateway, but they cannot communicate directly.

Configuration options

The following configuration key namespaces are currently supported for the `macvlan` network type:

- `maas` (MAAS network identification)
- `user` (free-form key/value for user metadata)

Note

LXD uses the CIDR notation[264] where network subnet information is required, for example, `192.0.2.0/24` or `2001:db8::/32`. This does not apply to cases where a single address is required, for example, local/remote addresses of tunnels, NAT addresses or specific addresses to apply to an instance.

The following configuration options are available for the `macvlan` network type: `gvrp` Whether to use GARP VLAN Registration Protocol (page 594)

Key:	`gvrp`
Type:	bool
Default:	`false`
Scope:	global

This option specifies whether to register the VLAN using the GARP VLAN Registration Protocol.

`maas.subnet.ipv4` MAAS IPv4 subnet to register instances in (page 594)

Key:	`maas.subnet.ipv4`
Type:	string
Condition:	IPv4 address; using the `network` property on the NIC
Scope:	global

`maas.subnet.ipv6` MAAS IPv6 subnet to register instances in (page 594)

Key:	`maas.subnet.ipv6`
Type:	string
Condition:	IPv4 address; using the `network` property on the NIC
Scope:	global

`mtu` MTU of the new interface (page 594)

[264] https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing

Key:	`mtu`
Type:	integer
Scope:	global

`parent` Parent interface to create `macvlan` NICs on (page 595)

Key:	`parent`
Type:	string
Scope:	local

`user.*` User-provided free-form key/value pairs (page 595)

Key:	`user.*`
Type:	string
Scope:	global

`vlan` VLAN ID to attach to (page 595)

Key:	`vlan`
Type:	integer
Scope:	global

Physical network

The `physical` network type connects to an existing physical network, which can be a network interface or a bridge, and serves as an uplink network for OVN.

This network type allows to specify presets to use when connecting OVN networks to a parent interface or to allow an instance to use a physical interface as a NIC. In this case, the instance NICs can simply set the `network`option to the network they connect to without knowing any of the underlying configuration details.

Configuration options

The following configuration key namespaces are currently supported for the `physical` network type:

- `bgp` (BGP peer configuration)
- `dns` (DNS server and resolution configuration)
- `ipv4` (L3 IPv4 configuration)
- `ipv6` (L3 IPv6 configuration)
- `maas` (MAAS network identification)
- `ovn` (OVN configuration)
- `user` (free-form key/value for user metadata)

> **Note**
>
> LXD uses the CIDR notation[265] where network subnet information is required, for example, `192.0.2.0/24` or `2001:db8::/32`. This does not apply to cases where a single address is required, for example, local/remote addresses of tunnels, NAT addresses or specific addresses to apply to an instance.

The following configuration options are available for the `physical` network type: `bgp.peers.NAME.address` Peer address for use by `ovn` downstream networks (page 596)

Key:	`bgp.peers.NAME.address`
Type:	string
Condition:	BGP server
Scope:	global

The address can be IPv4 or IPv6.

`bgp.peers.NAME.asn` Peer AS number for use by `ovn` downstream networks (page 596)

Key:	`bgp.peers.NAME.asn`
Type:	integer
Condition:	BGP server
Scope:	global

`bgp.peers.NAME.holdtime` Peer session hold time (page 596)

Key:	`bgp.peers.NAME.holdtime`
Type:	integer
Default:	`180`
Condition:	BGP server
Required:	no
Scope:	global

Specify the peer session hold time in seconds.

`bgp.peers.NAME.password` Peer session password for use by `ovn` downstream networks (page 596)

[265] https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing

Key:	`bgp.peers.NAME.password`
Type:	string
Default:	(no password)
Condition:	BGP server
Required:	no
Scope:	global

`dns.nameservers` DNS server IPs on physical network (page 597)

Key:	`dns.nameservers`
Type:	string
Condition:	standard mode
Scope:	global

Specify a list of DNS server IPs.

`gvrp` Whether to use GARP VLAN Registration Protocol (page 597)

Key:	`gvrp`
Type:	bool
Default:	`false`
Scope:	global

This option specifies whether to register the VLAN using the GARP VLAN Registration Protocol.

`ipv4.gateway` IPv4 address for the gateway and network (page 597)

Key:	`ipv4.gateway`
Type:	string
Condition:	standard mode
Scope:	global

Use CIDR notation.

`ipv4.ovn.ranges` IPv4 ranges to use for child OVN network routers (page 597)

Key:	`ipv4.ovn.ranges`
Type:	string
Scope:	global

Specify a comma-separated list of IPv4 ranges in FIRST-LAST format.

`ipv4.routes` Additional IPv4 CIDR subnets (page 597)

Key:	`ipv4.routes`
Type:	string
Condition:	IPv4 address
Scope:	global

Specify a comma-separated list of IPv4 CIDR subnets that can be used with child OVN network forwarders, load-balancers and *ipv4.routes.external* (page 464) setting.

`ipv4.routes.anycast` Whether to allow IPv4 routes on multiple networks/NICs (page 598)

Key:	`ipv4.routes.anycast`
Type:	bool
Default:	`false`
Condition:	IPv4 address
Scope:	global

If set to `true`, this option allows the overlapping routes to be used on multiple networks/NICs at the same time.

`ipv6.gateway` IPv6 address for the gateway and network (page 598)

Key:	`ipv6.gateway`
Type:	string
Condition:	standard mode
Scope:	global

Use CIDR notation.

`ipv6.ovn.ranges` IPv6 ranges to use for child OVN network routers (page 598)

Key:	`ipv6.ovn.ranges`
Type:	string
Scope:	global

Specify a comma-separated list of IPv6 ranges in FIRST-LAST format.

`ipv6.routes` Additional IPv6 CIDR subnets (page 598)

Key:	`ipv6.routes`
Type:	string
Condition:	IPv6 address
Scope:	global

Specify a comma-separated list of IPv6 CIDR subnets that can be used with child OVN network forwarders, load-balancers and *ipv6.routes.external* (page 465) setting.

`ipv6.routes.anycast` Whether to allow IPv6 routes on multiple networks/NICs (page 598)

Key:	`ipv6.routes.anycast`
Type:	bool
Default:	`false`
Condition:	IPv6 address
Scope:	global

If set to `true`, this option allows the overlapping routes to be used on multiple networks/NICs at the same time.

`maas.subnet.ipv4` MAAS IPv4 subnet to register instances in (page 599)

Key:	`maas.subnet.ipv4`
Type:	string
Condition:	IPv4 address; using the `network` property on the NIC
Scope:	global

`maas.subnet.ipv6` MAAS IPv6 subnet to register instances in (page 599)

Key:	`maas.subnet.ipv6`
Type:	string
Condition:	IPv6 address; using the `network` property on the NIC
Scope:	global

`mtu` MTU of the new interface (page 599)

Key:	`mtu`
Type:	integer
Scope:	global

`ovn.ingress_mode` How OVN NIC external IPs are advertised on uplink network (page 599)

Key:	`ovn.ingress_mode`
Type:	string
Default:	`l2proxy`
Condition:	standard mode
Scope:	global

Possible values are `l2proxy` (proxy ARP/NDP) and `routed`.

`parent` Existing interface to use for network (page 599)

Key:	`parent`
Type:	string
Scope:	local

`user.*` User-provided free-form key/value pairs (page 599)

Key:	`user.*`
Type:	string
Scope:	global

`vlan` VLAN ID to attach to (page 600)

Key:	`vlan`
Type:	integer
Scope:	global

Supported features

The following features are supported for the `physical` network type:

- *How to configure LXD as a BGP server* (page 214)

SR-IOV network

SR-IOV is a hardware standard that allows a single network card port to appear as several virtual network interfaces in a virtualized environment.

The `sriov` network type allows to specify presets to use when connecting instances to a parent interface. In this case, the instance NICs can simply set the `network` option to the network they connect to without knowing any of the underlying configuration details.

Configuration options

The following configuration key namespaces are currently supported for the `sriov` network type:

- `maas` (MAAS network identification)
- `user` (free-form key/value for user metadata)

> **Note**
>
> LXD uses the CIDR notation[266] where network subnet information is required, for example, `192.0.2.0/24` or `2001:db8::/32`. This does not apply to cases where a single address is required, for example, local/remote addresses of tunnels, NAT addresses or specific addresses to apply to an instance.

The following configuration options are available for the `sriov` network type: `maas.subnet.ipv4` MAAS IPv4 subnet to register instances in (page 600)

[266] https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing

Key:	`maas.subnet.ipv4`
Type:	string
Condition:	IPv4 address; using the `network` property on the NIC
Scope:	global

`maas.subnet.ipv6` MAAS IPv6 subnet to register instances in (page 601)

Key:	`maas.subnet.ipv6`
Type:	string
Condition:	IPv6 address; using the `network` property on the NIC
Scope:	global

`mtu` MTU of the new interface (page 601)

Key:	`mtu`
Type:	integer
Scope:	global

`parent` Parent interface to create `sriov` NICs on (page 601)

Key:	`parent`
Type:	string
Scope:	local

`user.*` User-provided free-form key/value pairs (page 601)

Key:	`user.*`
Type:	string
Scope:	global

`vlan` VLAN ID to attach to (page 601)

Key:	`vlan`
Type:	integer
Scope:	global

Related topics

How-to guides:

- *Networking* (page 210)

Explanation:

- *Networking setups* (page 353)

4.2.8. Cluster member configuration

Each cluster member has its own key/value configuration with the following supported namespaces:

- `user` (free form key/value for user metadata)
- `scheduler` (options related to how the member is automatically targeted by the cluster)

The following keys are currently supported: `scheduler.instance` Controls how instances are scheduled to run on this member (page 602)

Key:	`scheduler.instance`
Type:	string
Default:	`all`

Possible values are `all`, `manual`, and `group`. See *Automatic placement of instances* (page 372) for more information.

`user.*` Free form user key/value storage (page 602)

Key:	`user.*`
Type:	string

User keys can be used in search.

Related topics

How-to guides:

- *Clustering* (page 280)

Explanation:

- *Clusters* (page 370)

4.3. Production setup

Once you are ready for production, make sure your LXD server is configured to support the required load. You should also regularly *monitor the server metrics* (page 301).

4.3.1. Server settings for a LXD production setup

To allow your LXD server to run a large number of instances, configure the following settings to avoid hitting server limits.

The `Value` column contains the suggested value for each parameter.

`/etc/security/limits.conf`

> **Note**
>
> For users of the snap, those limits are automatically raised.

Domain	Type	Item	Value	Default	Description
*	soft	nofile	1048576	unset	Maximum number of open files
*	hard	nofile	1048576	unset	Maximum number of open files
root	soft	nofile	1048576	unset	Maximum number of open files
root	hard	nofile	1048576	unset	Maximum number of open files
*	soft	memlock	unlimited	unset	Maximum locked-in-memory address space (KB)
*	hard	memlock	unlimited	unset	Maximum locked-in-memory address space (KB)
root	soft	memlock	unlimited	unset	Maximum locked-in-memory address space (KB), only need with bpf syscall supervision
root	hard	memlock	unlimited	unset	Maximum locked-in-memory address space (KB), only need with bpf syscall supervision

/etc/sysctl.conf

Note

Reboot the server after changing any of these parameters.

fs.aio-max-nr Maximum number of concurrent asynchronous I/O operations (page 603)

Key:	fs.aio-max-nr
Type:	integer
Default:	65536

Suggested value: 524288

You might need to increase this limit further if you have a lot of workloads that use the AIO subsystem (for example, MySQL).

fs.inotify.max_queued_events Upper limit on the number of events that can be queued (page 603)

Key:	fs.inotify.max_queued_events
Type:	integer
Default:	16384

Suggested value: `1048576`

This option specifies the maximum number of events that can be queued to the corresponding `inotify` instance (see `inotify`[267] for more information).

`fs.inotify.max_user_instances` Upper limit on the number of `inotify` instances (page 604)

Key:	`fs.inotify.max_user_instances`
Type:	integer
Default:	`128`

Suggested value: `1048576`

This option specifies the maximum number of `inotify` instances that can be created per real user ID (see `inotify`[268] for more information).

`fs.inotify.max_user_watches` Upper limit on the number of watches (page 604)

Key:	`fs.inotify.max_user_watches`
Type:	integer
Default:	`8192`

Suggested value: `1048576`

This option specifies the maximum number of watches that can be created per real user ID (see `inotify`[269] for more information).

`kernel.dmesg_restrict` Whether to deny access to the messages in the kernel ring buffer (page 604)

Key:	`kernel.dmesg_restrict`
Type:	integer
Default:	`0`

Suggested value: `1`

Set this option to `1` to deny container access to the messages in the kernel ring buffer. Note that setting this value to `1` will also deny access to non-root users on the host system.

`kernel.keys.maxbytes` Maximum size of the key ring that non-root users can use (page 604)

Key:	`kernel.keys.maxbytes`
Type:	integer
Default:	`20000`

[267] https://man7.org/linux/man-pages/man7/inotify.7.html
[268] https://man7.org/linux/man-pages/man7/inotify.7.html
[269] https://man7.org/linux/man-pages/man7/inotify.7.html

Suggested value: `2000000`

`kernel.keys.maxkeys` Maximum number of keys that a non-root user can use (page 605)

Key:	`kernel.keys.maxkeys`
Type:	integer
Default:	`200`

Suggested value: `2000`

Set this option to a value that is higher than the number of instances.

`net.core.bpf_jit_limit` Limit on the size of eBPF JIT allocations (page 605)

Key:	`net.core.bpf_jit_limit`
Type:	integer
Default:	varies

Suggested value: `1000000000`

On kernels < 5.15 that are compiled with `CONFIG_BPF_JIT_ALWAYS_ON=y`, this value might limit the amount of instances that can be created.

`net.ipv4.neigh.default.gc_thresh3` Maximum number of entries in the IPv4 ARP table (page 605)

Key:	`net.ipv4.neigh.default.gc_thresh3`
Type:	integer
Default:	`1024`

Suggested value: `8192`

Increase this value if you plan to create over 1024 instances. Otherwise, you will get the error `neighbour: ndisc_cache: neighbor table overflow!` when the ARP table gets full and the instances cannot get a network configuration. See `ip-sysctl`[270] for more information.

`net.ipv6.neigh.default.gc_thresh3` Maximum number of entries in IPv6 ARP table (page 605)

Key:	`net.ipv6.neigh.default.gc_thresh3`
Type:	integer
Default:	`1024`

Suggested value: `8192`

[270] https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt

Increase this value if you plan to create over 1024 instances. Otherwise, you will get the error `neighbour: ndisc_cache: neighbor table overflow!` when the ARP table gets full and the instances cannot get a network configuration. See `ip-sysctl`[271] for more information.

`vm.max_map_count` Maximum number of memory map areas a process may have (page 606)

Key:	`vm.max_map_count`
Type:	integer
Default:	`65530`

Suggested value: `262144`

Memory map areas are used as a side-effect of calling `malloc`, directly by `mmap` and `mprotect`, and also when loading shared libraries.

Related topics

How-to guides:

- *How to benchmark performance* (page 297)
- *How to increase the network bandwidth* (page 299)
- *How to monitor metrics* (page 301)

Explanation:

- *Performance tuning* (page 375)

4.3.2. Provided metrics

LXD provides a number of instance metrics and internal metrics. See *How to monitor metrics* (page 301) for instructions on how to work with these metrics.

Instance metrics

The following instance metrics are provided:

Metric	Description
lxd_cpu_effective_total	Total number of effective CPUs
lxd_cpu_seconds_total{cpu="<cpu>", mode="<mode>"}	Total number of CPU time used (in sec
lxd_disk_read_bytes_total{device="<dev>"}	Total number of bytes read
lxd_disk_reads_completed_total{device="<dev>"}	Total number of completed reads
lxd_disk_written_bytes_total{device="<dev>"}	Total number of bytes written
lxd_disk_writes_completed_total{device="<dev>"}	Total number of completed writes
lxd_filesystem_avail_bytes{device="<dev>",fstype="<type>"}	Available space (in bytes)
lxd_filesystem_free_bytes{device="<dev>",fstype="<type>"}	Free space (in bytes)
lxd_filesystem_size_bytes{device="<dev>",fstype="<type>"}	Size of the file system (in bytes)
lxd_memory_Active_anon_bytes	Amount of anonymous memory on act
lxd_memory_Active_bytes	Amount of memory on active LRU list
lxd_memory_Active_file_bytes	Amount of file-backed memory on acti

co

[271] https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt

Table 1 – continued from previous page

Metric	Description
lxd_memory_Cached_bytes	Amount of cached memory
lxd_memory_Dirty_bytes	Amount of memory waiting to be writt
lxd_memory_HugepagesFree_bytes	Amount of free memory for hugetlb
lxd_memory_HugepagesTotal_bytes	Amount of used memory for hugetlb
lxd_memory_Inactive_anon_bytes	Amount of anonymous memory on ina
lxd_memory_Inactive_bytes	Amount of memory on inactive LRU lis
lxd_memory_Inactive_file_bytes	Amount of file-backed memory on inac
lxd_memory_Mapped_bytes	Amount of mapped memory
lxd_memory_MemAvailable_bytes	Amount of available memory
lxd_memory_MemFree_bytes	Amount of free memory
lxd_memory_MemTotal_bytes	Amount of used memory
lxd_memory_OOM_kills_total	The number of out-of-memory kills
lxd_memory_RSS_bytes	Amount of anonymous and swap cache
lxd_memory_Shmem_bytes	Amount of cached file system data tha
lxd_memory_Swap_bytes	Amount of used swap memory
lxd_memory_Unevictable_bytes	Amount of unevictable memory
lxd_memory_Writeback_bytes	Amount of memory queued for syncing
lxd_network_receive_bytes_total{device="<dev>"}	Amount of received bytes on a given in
lxd_network_receive_drop_total{device="<dev>"}	Amount of received dropped bytes on
lxd_network_receive_errs_total{device="<dev>"}	Amount of received errors on a given i
lxd_network_receive_packets_total{device="<dev>"}	Amount of received packets on a given
lxd_network_transmit_bytes_total{device="<dev>"}	Amount of transmitted bytes on a give
lxd_network_transmit_drop_total{device="<dev>"}	Amount of transmitted dropped bytes
lxd_network_transmit_errs_total{device="<dev>"}	Amount of transmitted errors on a give
lxd_network_transmit_packets_total{device="<dev>"}	Amount of transmitted packets on a gi
lxd_procs_total	Number of running processes

Internal metrics

The following internal metrics are provided:

Metric	Description
lxd_api_requests_completed	Total number of completed requests. See *API rates metrics* (page 608).
lxd_api_requests_ongoing	Number of requests currently being handled. See *API rates metrics* (page 608).
lxd_go_alloc_bytes_total	Total number of bytes allocated (even if freed)
lxd_go_alloc_bytes	Number of bytes allocated and still in use
lxd_go_buck_hash_sys_bytes	Number of bytes used by the profiling bucket hash table
lxd_go_frees_total	Total number of frees
lxd_go_gc_sys_bytes	Number of bytes used for garbage collection system metadata
lxd_go_goroutines	Number of goroutines that currently exist
lxd_go_heap_alloc_bytes	Number of heap bytes allocated and still in use
lxd_go_heap_idle_bytes	Number of heap bytes waiting to be used
lxd_go_heap_inuse_bytes	Number of heap bytes that are in use
lxd_go_heap_objects	Number of allocated objects
lxd_go_heap_released_bytes	Number of heap bytes released to OS
lxd_go_heap_sys_bytes	Number of heap bytes obtained from system
lxd_go_lookups_total	Total number of pointer lookups
lxd_go_mallocs_total	Total number of `mallocs`
lxd_go_mcache_inuse_bytes	Number of bytes in use by `mcache` structures
lxd_go_mcache_sys_bytes	Number of bytes used for `mcache` structures obtained from system
lxd_go_mspan_inuse_bytes	Number of bytes in use by `mspan` structures
lxd_go_mspan_sys_bytes	Number of bytes used for `mspan` structures obtained from system
lxd_go_next_gc_bytes	Number of heap bytes when next garbage collection will take place
lxd_go_other_sys_bytes	Number of bytes used for other system allocations
lxd_go_stack_inuse_bytes	Number of bytes in use by the stack allocator
lxd_go_stack_sys_bytes	Number of bytes obtained from system for stack allocator
lxd_go_sys_bytes	Number of bytes obtained from system
lxd_operations_total	Number of running operations
lxd_uptime_seconds	Daemon uptime (in seconds)
lxd_warnings_total	Number of active warnings

API rates metrics

The API rates metrics include `lxd_api_requests_completed_total` and `lxd_api_requests_ongoing`. These metrics can be consumed by an observability tool deployed externally (for example, the Canonical Observability Stack[272] or another third-party tool) to help identify failures or overload on a LXD server. You can set thresholds on the observability tools for these metrics' values to trigger alarms and take programmatic actions.

These metrics consider all endpoints in the *LXD REST API* (page 623), with the exception of the / endpoint. Requests using an invalid URL are also counted. Requests against the metrics server are also counted. Both introduced metrics include a label `entity_type` based on the main entity type that the endpoint is operating on.

[272] https://charmhub.io/topics/canonical-observability-stack

`lxd_api_requests_ongoing` contains the number of requests that are not yet completed by the time the metrics are queried. A request is considered completed when the response is returned to the client and any asynchronous operations spawned by that request are done. `lxd_api_requests_completed_total` contains the number of completed requests. This metric includes an additional label named `result` based on the outcome of the request. The label can have one of the following values:

- `error_server`, for errors on the server side, this includes responses with HTTP status codes from 500 to 599. Any failed asynchronous operations also fall into this category.
- `error_client`, for responses with HTTP status codes from 400 to 499, indicating an error on the client side.
- `succeeded`, for endpoints that executed successfully.

Related topics

How-to guides:

- *How to monitor metrics* (page 301)

Explanation:

- *Performance tuning* (page 375)

4.4. Fine-grained permissions

If you are managing user access via *Fine-grained authorization* (page 364), check which *permissions* (page 365) can be assigned to groups.

4.4.1. Permissions

When managing user access via *Fine-grained authorization* (page 364), you add identities to groups and then grant entitlements against specific LXD API resources to these groups.

Each LXD API resource has a particular entity type, and each entity type has a set of entitlements that can be granted against API resources of that type.

Below is a description of each entity type, and a list of entitlements that can be granted against entities of that type.

Server

Entity type name: `server`

The `server` entity type is the top-level entity type for the LXD system. Entitlements that are granted at this level might cascade to projects and other resources:

`admin`
: Grants full access to LXD as if via Unix socket.

`viewer`
: Grants access to view all resources in the LXD server.

`can_edit`
: Grants permission to edit server configuration, to edit cluster member configuration, to update the state of a cluster member, to create, edit, and delete cluster groups, to update cluster member certificates, and to edit or delete warnings.

`permission_manager`
: Grants permission to view permissions, to create, edit, and delete identities, to view,

create, edit, and delete authorization groups, and to view, create, edit, and delete identity provider groups. Note that clients with this permission are able to elevate their own privileges.

`can_view_permissions`
: Grants permission to view permissions.

`can_create_identities`
: Grants permission to create identities.

`can_view_identities`
: Grants permission to view identities.

`can_edit_identities`
: Grants permission to edit identities.

`can_delete_identities`
: Grants permission to delete identities.

`can_create_groups`
: Grants permission to create authorization groups.

`can_view_groups`
: Grants permission to view authorization groups.

`can_edit_groups`
: Grants permission to edit authorization groups.

`can_delete_groups`
: Grants permission to delete authorization groups.

`can_create_identity_provider_groups`
: Grants permission to create identity provider groups.

`can_view_identity_provider_groups`
: Grants permission to view identity provider groups.

`can_edit_identity_provider_groups`
: Grants permission to edit identity provider groups.

`can_delete_identity_provider_groups`
: Grants permission to delete identity provider groups.

`storage_pool_manager`
: Grants permission to create, edit, and delete storage pools.

`can_create_storage_pools`
: Grants permission to create storage pools.

`can_edit_storage_pools`
: Grants permission to edit storage pools.

`can_delete_storage_pools`
: Grants permission to delete storage pools.

`project_manager`
: Grants permission to view, create, edit, and delete projects, and to create, edit, and delete any resources that are owned by those projects.

`can_create_projects`
: Grants permission to create projects.

`can_view_projects`
: Grants permission to view projects, and all resources within those projects.

`can_edit_projects`
: Grants permission to edit projects, and all resources within those projects.

`can_delete_projects`
: Grants permission to delete projects.

`can_override_cluster_target_restriction`
: If a project is configured with `restricted.cluster.target`, clients with this permission can override the restriction.

`can_view_privileged_events`
: Grants permission to view privileged event types, such as logging events.

`can_view_resources`
: Grants permission to view server and storage pool resource usage information.

`can_view_metrics`
: Grants permission to view all server and project level metrics.

`can_view_warnings`
: Grants permission to view warnings.

`can_view_unmanaged_networks`
: Grants permission to view unmanaged networks on the LXD host machines.

Project

Entity type name: `project`

Entitlements that are granted at the `project` level might cascade to project specific resources (such as instances):

`operator`
: Grants permission to create, view, edit, and delete all resources belonging to the project, but does not grant permission to edit the project configuration itself.

`viewer`
: Grants permission to view all resources belonging to the project.

`can_view`
: Grants permission to view the project.

`can_edit`
: Grants permission to edit the project.

`can_delete`
: Grants permission to delete the project.

`image_manager`
: Grants permission to create, view, edit, and delete all images belonging to the project.

`can_create_images`
: Grants permission to create images.

`can_view_images`
: Grants permission to view images.

`can_edit_images`
: Grants permission to edit images.

`can_delete_images`
: Grants permission to delete images.

`image_alias_manager`
: Grants permission to create, view, edit, and delete all image aliases belonging to the project.

`can_create_image_aliases`
: Grants permission to create image aliases.

`can_view_image_aliases`
: Grants permission to view image aliases.

`can_edit_image_aliases`
: Grants permission to edit image aliases.

`can_delete_image_aliases`
: Grants permission to delete image aliases.

`instance_manager`
: Grants permission to create, view, edit, and delete all instances belonging to the project.

`can_create_instances`
: Grants permission to create instances.

`can_view_instances`
: Grants permission to view instances.

`can_edit_instances`
: Grants permission to edit instances.

`can_delete_instances`
: Grants permission to delete instances.

`can_operate_instances`
: Grants permission to view instances, manage their state, manage their snapshots and backups, start terminal or console sessions, and access their files.

`network_manager`
: Grants permission to create, view, edit, and delete all networks belonging to the project.

`can_create_networks`
: Grants permission to create networks.

`can_view_networks`
: Grants permission to view networks.

`can_edit_networks`
: Grants permission to edit networks.

`can_delete_networks`
: Grants permission to delete networks.

`network_acl_manager`
: Grants permission to create, view, edit, and delete all network ACLs belonging to the

project.

`can_create_network_acls`
: Grants permission to create network ACLs.

`can_view_network_acls`
: Grants permission to view network ACLs.

`can_edit_network_acls`
: Grants permission to edit network ACLs.

`can_delete_network_acls`
: Grants permission to delete network ACLs.

`network_zone_manager`
: Grants permission to create, view, edit, and delete all network zones belonging to the project.

`can_create_network_zones`
: Grants permission to create network zones.

`can_view_network_zones`
: Grants permission to view network zones.

`can_edit_network_zones`
: Grants permission to edit network zones.

`can_delete_network_zones`
: Grants permission to delete network zones.

`profile_manager`
: Grants permission to create, view, edit, and delete all profiles belonging to the project.

`can_create_profiles`
: Grants permission to create profiles.

`can_view_profiles`
: Grants permission to view profiles.

`can_edit_profiles`
: Grants permission to edit profiles.

`can_delete_profiles`
: Grants permission to delete profiles.

`storage_volume_manager`
: Grants permission to create, view, edit, and delete all storage volumes belonging to the project.

`can_create_storage_volumes`
: Grants permission to create storage volumes.

`can_view_storage_volumes`
: Grants permission to view storage volumes.

`can_edit_storage_volumes`
: Grants permission to edit storage volumes.

`can_delete_storage_volumes`
: Grants permission to delete storage volumes.

`storage_bucket_manager`
: Grants permission to create, view, edit, and delete all storage buckets belonging to the project.

`can_create_storage_buckets`
: Grants permission to create storage buckets.

`can_view_storage_buckets`
: Grants permission to view storage buckets.

`can_edit_storage_buckets`
: Grants permission to edit storage buckets.

`can_delete_storage_buckets`
: Grants permission to delete storage buckets.

`can_view_operations`
: Grants permission to view operations relating to the project.

`can_view_events`
: Grants permission to view events relating to the project.

`can_view_metrics`
: Grants permission to view project level metrics.

Storage pool

Entity type name: `storage_pool`

`can_edit`
: Grants permission to edit the storage pool.

`can_delete`
: Grants permission to delete the storage pool.

Identity

Entity type name: `identity`

`can_view`
: Grants permission to view the identity.

`can_edit`
: Grants permission to edit the identity.

`can_delete`
: Grants permission to delete the identity.

Group

Entity type name: `group`

`can_view`
: Grants permission to view the group. Identities can always view groups that they are a member of.

`can_edit`
: Grants permission to edit the group.

can_delete
Grants permission to delete the group.

Identity provider group

Entity type name: `identity_provider_group`

can_view
Grants permission to view the identity provider group.

can_edit
Grants permission to edit the identity provider group.

can_delete
Grants permission to delete the identity provider group.

Certificate

Entity type name: `certificate`

can_view
Grants permission to view the certificate.

can_edit
Grants permission to edit the certificate.

can_delete
Grants permission to delete the certificate.

Instance

Entity type name: `instance`

user
Grants permission to view the instance, to access files, and to start a terminal or console session.

operator
Grants permission to view the instance, to access files, start a terminal or console session, and to manage snapshots and backups.

can_edit
Grants permission to edit the instance.

can_delete
Grants permission to delete the instance.

can_view
Grants permission to view the instance and any snapshots or backups it might have.

can_update_state
Grants permission to change the instance state.

can_manage_snapshots
Grants permission to create and delete snapshots of the instance.

can_manage_backups
Grants permission to create and delete backups of the instance.

can_connect_sftp
Grants permission to get an SFTP client for the instance.

can_access_files
Grants permission to push or pull files into or out of the instance.

can_access_console
Grants permission to start a console session.

can_exec
Grants permission to start a terminal session.

Image

Entity type name: `image`

can_edit
Grants permission to edit the image.

can_delete
Grants permission to delete the image.

can_view
Grants permission to view the image.

Image alias

Entity type name: `image_alias`

can_edit
Grants permission to edit the image alias.

can_delete
Grants permission to delete the image alias.

can_view
Grants permission to view the image alias.

Network

Entity type name: `network`

can_edit
Grants permission to edit the network.

can_delete
Grants permission to delete the network.

can_view
Grants permission to view the network.

Network ACL

Entity type name: `network_acl`

can_edit
Grants permission to edit the network ACL.

can_delete
: Grants permission to delete the network ACL.

can_view
: Grants permission to view the network ACL.

Network zone

Entity type name: `network_zone`

can_edit
: Grants permission to edit the network zone.

can_delete
: Grants permission to delete the network zone.

can_view
: Grants permission to view the network zone.

Profile

Entity type name: `profile`

can_edit
: Grants permission to edit the profile.

can_delete
: Grants permission to delete the profile.

can_view
: Grants permission to view the profile.

Storage volume

Entity type name: `storage_volume`

can_edit
: Grants permission to edit the storage volume.

can_delete
: Grants permission to delete the storage volume.

can_view
: Grants permission to view the storage volume and any snapshots or backups it might have.

can_manage_snapshots
: Grants permission to create and delete snapshots of the storage volume.

can_manage_backups
: Grants permission to create and delete backups of the storage volume.

Storage bucket

Entity type name: `storage_bucket`

can_edit
: Grants permission to edit the storage bucket.

can_delete
Grants permission to delete the storage bucket.

can_view
Grants permission to view the storage bucket.

4.5. REST API

All communication between LXD and its clients happens using a RESTful API over HTTP. Check the list of API extensions to see if a feature is available in your version of the API.

4.5.1. REST API

REST API

All communication between LXD and its clients happens using a RESTful API over HTTP. This API is encapsulated over either TLS (for remote operations) or a Unix socket (for local operations).

See *Remote API authentication* (page 358) for information about how to access the API remotely.

> **Tip**
>
> - For examples on how the API is used, run any command of the LXD client (*lxc* (page 690)) with the `--debug` flag. The debug information displays the API calls and the return values.
> - For quickly querying the API, the LXD client provides a *lxc query* (page 869) command.

API versioning

The list of supported major API versions can be retrieved using `GET /`.

The reason for a major API bump is if the API breaks backward compatibility.

Feature additions done without breaking backward compatibility only result in addition to `api_extensions` which can be used by the client to check if a given feature is supported by the server.

Return values

There are three standard return types:

- Standard return value
- Background operation
- Error

Standard return value

For a standard synchronous operation, the following JSON object is returned:

```
{
    "type": "sync",
```

(continues on next page)

(continued from previous page)

```
    "status": "Success",
    "status_code": 200,
    "metadata": {}                          // Extra resource/action specific
metadata
}
```

HTTP code must be 200.

Background operation

When a request results in a background operation, the HTTP code is set to 202 (Accepted) and the Location HTTP header is set to the operation URL.

The body is a JSON object with the following structure:

```
{
    "type": "async",
    "status": "OK",
    "status_code": 100,
    "operation": "/1.0/instances/<id>",                 // URL to the
background operation
    "metadata": {}                                      // Operation metadata
(see below)
}
```

The operation metadata structure looks like:

```
{
    "id": "a40f5541-5e98-454f-b3b6-8a51ef5dbd3c",       // UUID of the
operation
    "class": "websocket",                               // Class of the
operation (task, websocket or token)
    "created_at": "2015-11-17T22:32:02.226176091-05:00",    // When the operation
was created
    "updated_at": "2015-11-17T22:32:02.226176091-05:00",    // Last time the
operation was updated
    "status": "Running",                                // String version of
the operation's status
    "status_code": 103,                                 // Integer version of
the operation's status (use this rather than status)
    "resources": {                                      // Dictionary of
resource types (container, snapshots, images) and affected resources
      "containers": [
        "/1.0/instances/test"
      ]
    },
    "metadata": {                                       // Metadata specific
to the operation in question (in this case, exec)
      "fds": {
        "0": "2a4a97af81529f6608dca31f03a7b7e47acc0b8dc6514496eb25e325f9e4fa6a",
```

(continues on next page)

(continued from previous page)

```
        "control":
"5b64c661ef313b423b5317ba9cb6410e40b705806c28255f601c0ef603f079a7"
      }
    },
    "may_cancel": false,                                  // Whether the
operation can be canceled (DELETE over REST)
    "err": ""                                             // The error string
should the operation have failed
}
```

The body is mostly provided as a user friendly way of seeing what's going on without having to pull the target operation, all information in the body can also be retrieved from the background operation URL.

Error

There are various situations in which something may immediately go wrong, in those cases, the following return value is used:

```
{
    "type": "error",
    "error": "Failure",
    "error_code": 400,
    "metadata": {}                    // More details about the error
}
```

HTTP code must be one of of 400, 401, 403, 404, 409, 412 or 500.

Status codes

The LXD REST API often has to return status information, be that the reason for an error, the current state of an operation or the state of the various resources it exports.

To make it simple to debug, all of those are always doubled. There is a numeric representation of the state which is guaranteed never to change and can be relied on by API clients. Then there is a text version meant to make it easier for people manually using the API to figure out what's happening.

In most cases, those will be called status and `status_code`, the former being the user-friendly string representation and the latter the fixed numeric value.

The codes are always 3 digits, with the following ranges:

- 100 to 199: resource state (started, stopped, ready, …)
- 200 to 399: positive action result
- 400 to 599: negative action result
- 600 to 999: future use

List of current status codes

Code	Meaning
100	Operation created
101	Started
102	Stopped
103	Running
104	Canceling
105	Pending
106	Starting
107	Stopping
108	Aborting
109	Freezing
110	Frozen
111	Thawed
112	Error
113	Ready
200	Success
400	Failure
401	Canceled

Recursion

To optimize queries of large lists, recursion is implemented for collections. A `recursion` argument can be passed to a GET query against a collection.

The default value is 0 which means that collection member URLs are returned. Setting it to 1 will have those URLs be replaced by the object they point to (typically another JSON object).

Recursion is implemented by simply replacing any pointer to an job (URL) by the object itself.

Filtering

To filter your results on certain values, filter is implemented for collections. A `filter` argument can be passed to a GET query against a collection.

Filtering is available for the instance, image and storage volume endpoints.

There is no default value for filter which means that all results found will be returned. The following is the language used for the filter argument:

```
?filter=field_name eq desired_field_assignment
```

The language follows the OData conventions for structuring REST API filtering logic. Logical operators are also supported for filtering: not (`not`), equals (`eq`), not equals (`ne`), and (`and`), or (`or`). Filters are evaluated with left associativity. Values with spaces can be surrounded with quotes. Nesting filtering is also supported. For instance, to filter on a field in a configuration you would pass:

```
?filter=config.field_name eq desired_field_assignment
```

For filtering on device attributes you would pass:

```
?filter=devices.device_name.field_name eq desired_field_assignment
```

Here are a few GET query examples of the different filtering methods mentioned above:

```
containers?filter=name eq "my container" and status eq Running

containers?filter=config.image.os eq ubuntu or devices.eth0.nictype eq bridged

images?filter=Properties.os eq Centos and not UpdateSource.Protocol eq
simplestreams
```

Asynchronous operations

Any operation which may take more than a second to be done must be done in the background, returning a background operation ID to the client.

The client will then be able to either poll for a status update or wait for a notification using the long-poll API.

Notifications

A WebSocket-based API is available for notifications, different notification types exist to limit the traffic going to the client.

It's recommended that the client always subscribes to the operations notification type before triggering remote operations so that it doesn't have to then poll for their status.

PUT vs PATCH

The LXD API supports both PUT and PATCH to modify existing objects:

The PUT method

PUT *replaces* the entire object with a new definition. Since it overwrites the existing state, it's often called after retrieving and recording the current object state through GET.

To avoid race conditions, the ETag header should be read from the GET response and sent as If-Match for the PUT request. This will cause LXD to fail the request if the object was modified between GET and PUT.

The PATCH method

PATCH can be used to modify a single field inside an object by only specifying the property that you want to change. To unset a key, setting it to empty will usually do the trick, but there are cases where PATCH won't work and PUT needs to be used instead.

Instances, containers and virtual-machines

The documentation shows paths such as `/1.0/instances/...`, which were introduced with LXD 3.19. Older releases that supported only containers and not virtual machines supply the exact same API at `/1.0/containers/...`.

For backward compatibility reasons, LXD does still expose and support that `/1.0/containers` API, though for the sake of brevity, we decided not to double-document everything.

An additional endpoint at `/1.0/virtual-machines` is also present and much like `/1.0/containers` will only show you instances of that type.

API structure

LXD has an auto-generated Swagger[273] specification describing its API endpoints. The YAML version of this API specification can be found in `rest-api.yaml`[274]. See *Main API specification* (page 623) for a convenient web rendering of it.

Main API specification

API extensions

The changes below were introduced to the LXD API after the 1.0 API was finalized.

They are all backward compatible and can be detected by client tools by looking at the `api_extensions` field in `GET /1.0`.

`storage_zfs_remove_snapshots`

A *`zfs.remove_snapshots`* (page 570) daemon configuration key was introduced.

It's a Boolean that defaults to `false` and that when set to `true` instructs LXD to remove any needed snapshot when attempting to restore another.

This is needed as ZFS will only let you restore the latest snapshot.

`container_host_shutdown_timeout`

A *`boot.host_shutdown_timeout`* (page 419) container configuration key was introduced.

It's an integer which indicates how long LXD should wait for the container to stop before killing it.

Its value is only used on clean LXD daemon shutdown. It defaults to 30s.

`container_stop_priority`

A *`boot.stop.priority`* (page 419) container configuration key was introduced.

It's an integer which indicates the priority of a container during shutdown.

Containers will shutdown starting with the highest priority level.

Containers with the same priority will shutdown in parallel. It defaults to 0.

`container_syscall_filtering`

A number of new syscalls related container configuration keys were introduced.

- *`security.syscalls.deny_default`* (page 439)

[273] https://swagger.io/
[274] https://github.com/canonical/lxd/blob/main/doc/rest-api.yaml

- *security.syscalls.deny_compat* (page 439)
- *security.syscalls.deny* (page 438)
- *security.syscalls.allow* (page 438)

See *Instance configuration* (page 414) for how to use them.

> **Note**
>
> Initially, those configuration keys were (accidentally) introduced with offensive names. They have since been renamed (`container_syscall_filtering_allow_deny_syntax`), and the old names are no longer accepted.

auth_pki

This indicates support for PKI authentication mode.

In this mode, the client and server both must use certificates issued by the same PKI.

See *Security* (page 376) for details.

container_last_used_at

A `last_used_at` field was added to the `GET /1.0/containers/<name>` endpoint.

It is a timestamp of the last time the container was started.

If a container has been created but not started yet, `last_used_at` field will be `1970-01-01T00:00:00Z`

etag

Add support for the ETag header on all relevant endpoints.

This adds the following HTTP header on answers to GET:

- ETag (SHA-256 of user modifiable content)

And adds support for the following HTTP header on PUT requests:

- If-Match (ETag value retrieved through previous GET)

This makes it possible to GET a LXD object, modify it and PUT it without risking to hit a race condition where LXD or another client modified the object in the meantime.

patch

Add support for the HTTP PATCH method.

PATCH allows for partial update of an object in place of PUT.

usb_devices

Add support for USB hotplug.

https_allowed_credentials

To use LXD API with all Web Browsers (via SPAs) you must send credentials (certificate) with each XHR (in order for this to happen, you should set `withCredentials=true`[275] flag to each XHR Request).

Some browsers like Firefox and Safari can't accept server response without `Access-Control-Allow-Credentials: true` header. To ensure that the server will return a response with that header, set *core.https_allowed_credentials* (page 402) to `true`.

image_compression_algorithm

This adds support for a `compression_algorithm` property when creating an image (`POST /1.0/images`).

Setting this property overrides the server default value (*images.compression_algorithm* (page 409)).

directory_manipulation

This allows for creating and listing directories via the LXD API, and exports the file type via the X-LXD-type header, which can be either `file` or `directory` right now.

container_cpu_time

This adds support for retrieving CPU time for a running container.

storage_zfs_use_refquota

Introduces a new server property *zfs.use_refquota* (page 570) which instructs LXD to set the `refquota` property instead of `quota` when setting a size limit on a container. LXD will also then use `usedbydataset` in place of `used` when being queried about disk utilization.

This effectively controls whether disk usage by snapshots should be considered as part of the container's disk space usage.

storage_lvm_mount_options

Adds a new `storage.lvm_mount_options` daemon configuration option which defaults to `discard` and allows the user to set addition mount options for the file system used by the LVM LV.

[275] https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/withCredentials

`network`

Network management API for LXD.

This includes:

- Addition of the `managed` property on `/1.0/networks` entries
- All the network configuration options (see *Network configuration* (page 210) for details)
- `POST /1.0/networks` (see *RESTful API* (page 618) for details)
- `PUT /1.0/networks/<entry>` (see *RESTful API* (page 618) for details)
- `PATCH /1.0/networks/<entry>` (see *RESTful API* (page 618) for details)
- `DELETE /1.0/networks/<entry>` (see *RESTful API* (page 618) for details)
- `ipv4.address` property on `nic` type devices (when `nictype` is `bridged`)
- `ipv6.address` property on `nic` type devices (when `nictype` is `bridged`)
- `security.mac_filtering` property on `nic` type devices (when `nictype` is `bridged`)

`profile_usedby`

Adds a new `used_by` field to profile entries listing the containers that are using it.

`container_push`

When a container is created in push mode, the client serves as a proxy between the source and target server. This is useful in cases where the target server is behind a NAT or firewall and cannot directly communicate with the source server and operate in pull mode.

`container_exec_recording`

Introduces a new Boolean `record-output`, parameter to `/1.0/containers/<name>/exec` which when set to `true` and combined with with `wait-for-websocket` set to `false`, will record stdout and stderr to disk and make them available through the logs interface.

The URL to the recorded output is included in the operation metadata once the command is done running.

That output will expire similarly to other log files, typically after 48 hours.

`certificate_update`

Adds the following to the REST API:

- ETag header on GET of a certificate
- PUT of certificate entries
- PATCH of certificate entries

`container_exec_signal_handling`

Adds support `/1.0/containers/<name>/exec` for forwarding signals sent to the client to the processes executing in the container. Currently SIGTERM and SIGHUP are forwarded. Further signals that can be forwarded might be added later.

`gpu_devices`

Enables adding GPUs to a container.

`container_image_properties`

Introduces a new `image` configuration key space. Read-only, includes the properties of the parent image.

`migration_progress`

Transfer progress is now exported as part of the operation, on both sending and receiving ends. This shows up as a `fs_progress` attribute in the operation metadata.

`id_map`

Enables setting the *security.idmap.isolated* (page 436), *security.idmap.size* (page 436), and *raw.idmap* (page 431) fields.

`network_firewall_filtering`

Add two new keys, *ipv4.firewall* (page 579) and *ipv6.firewall* (page 582) which if set to `false` will turn off the generation of `iptables` FORWARDING rules. NAT rules will still be added so long as the matching *ipv4.nat* (page 580) or *ipv6.nat* (page 582) key is set to `true`.

Rules necessary for `dnsmasq` to work (DHCP/DNS) will always be applied if `dnsmasq` is enabled on the bridge.

`network_routes`

Introduces *ipv4.routes* (page 580) and *ipv6.routes* (page 583) which allow routing additional subnets to a LXD bridge.

`storage`

Storage management API for LXD.

This includes:

- `GET /1.0/storage-pools`
- `POST /1.0/storage-pools` (see *RESTful API* (page 618) for details)
- `GET /1.0/storage-pools/<name>` (see *RESTful API* (page 618) for details)
- `POST /1.0/storage-pools/<name>` (see *RESTful API* (page 618) for details)
- `PUT /1.0/storage-pools/<name>` (see *RESTful API* (page 618) for details)
- `PATCH /1.0/storage-pools/<name>` (see *RESTful API* (page 618) for details)

- DELETE /1.0/storage-pools/<name> (see *RESTful API* (page 618) for details)
- GET /1.0/storage-pools/<name>/volumes (see *RESTful API* (page 618) for details)
- GET /1.0/storage-pools/<name>/volumes/<volume_type> (see *RESTful API* (page 618) for details)
- POST /1.0/storage-pools/<name>/volumes/<volume_type> (see *RESTful API* (page 618) for details)
- GET /1.0/storage-pools/<pool>/volumes/<volume_type>/<name> (see *RESTful API* (page 618) for details)
- POST /1.0/storage-pools/<pool>/volumes/<volume_type>/<name> (see *RESTful API* (page 618) for details)
- PUT /1.0/storage-pools/<pool>/volumes/<volume_type>/<name> (see *RESTful API* (page 618) for details)
- PATCH /1.0/storage-pools/<pool>/volumes/<volume_type>/<name> (see *RESTful API* (page 618) for details)
- DELETE /1.0/storage-pools/<pool>/volumes/<volume_type>/<name> (see *RESTful API* (page 618) for details)
- All storage configuration options (see *Storage configuration* (page 175) for details)

file_delete

Implements DELETE in /1.0/containers/<name>/files

file_append

Implements the X-LXD-write header which can be one of overwrite or append.

network_dhcp_expiry

Introduces *ipv4.dhcp.expiry* (page 579) and *ipv6.dhcp.expiry* (page 581) allowing to set the DHCP lease expiry time.

storage_lvm_vg_rename

Introduces the ability to rename a volume group by setting *lvm.vg_name* (page 559).

storage_lvm_thinpool_rename

Introduces the ability to rename a thin pool name by setting *lvm.thinpool_name* (page 558).

network_vlan

This adds a new *vlan* (page 457) property to macvlan network devices.

When set, this will instruct LXD to attach to the specified VLAN. LXD will look for an existing interface for that VLAN on the host. If one can't be found it will create one itself and then use that as the macvlan parent.

`image_create_aliases`

Adds a new `aliases` field to `POST /1.0/images` allowing for aliases to be set at image creation/import time.

`container_stateless_copy`

This introduces a new `live` attribute in `POST /1.0/containers/<name>`. Setting it to `false` tells LXD not to attempt running state transfer.

`container_only_migration`

Introduces a new Boolean `container_only` attribute. When set to `true` only the container will be copied or moved.

`storage_zfs_clone_copy`

Introduces a new Boolean *`zfs.clone_copy`* (page 566) property for ZFS storage pools. When set to `false` copying a container will be done through `zfs send` and receive. This will make the target container independent of its source container thus avoiding the need to keep dependent snapshots in the ZFS pool around. However, this also entails less efficient storage usage for the affected pool. The default value for this property is `true`, i.e. space-efficient snapshots will be used unless explicitly set to `false`.

`unix_device_rename`

Introduces the ability to rename the `unix-block`/`unix-char` device inside container by setting `path`, and the `source` attribute is added to specify the device on host. If `source` is set without a `path`, we should assume that `path` will be the same as `source`. If `path` is set without `source` and `major`/`minor` isn't set, we should assume that `source` will be the same as `path`. So at least one of them must be set.

`storage_rsync_bwlimit`

When `rsync` has to be invoked to transfer storage entities setting `rsync.bwlimit` places an upper limit on the amount of socket I/O allowed.

`network_vxlan_interface`

This introduces a new *`tunnel.NAME.interface`* (page 585) option for networks.

This key control what host network interface is used for a VXLAN tunnel.

`storage_btrfs_mount_options`

This introduces the *`btrfs.mount_options`* (page 523) property for Btrfs storage pools.

This key controls what mount options will be used for the Btrfs storage pool.

entity_description

This adds descriptions to entities like containers, snapshots, networks, storage pools and volumes.

image_force_refresh

This allows forcing a refresh for an existing image.

storage_lvm_lv_resizing

This introduces the ability to resize logical volumes by setting the `size` property in the containers root disk device.

id_map_base

This introduces a new *security.idmap.base* (page 435) allowing the user to skip the map auto-selection process for isolated containers and specify what host UID/GID to use as the base.

file_symlinks

This adds support for transferring symlinks through the file API. X-LXD-type can now be `symlink` with the request content being the target path.

container_push_target

This adds the `target` field to `POST /1.0/containers/<name>` which can be used to have the source LXD host connect to the target during migration.

network_vlan_physical

Allows use of *vlan* (page 600) property with `physical` network devices.

When set, this will instruct LXD to attach to the specified VLAN on the `parent` interface. LXD will look for an existing interface for that `parent` and VLAN on the host. If one can't be found it will create one itself. Then, LXD will directly attach this interface to the container.

storage_images_delete

This enabled the storage API to delete storage volumes for images from a specific storage pool.

container_edit_metadata

This adds support for editing a container `metadata.yaml` and related templates via API, by accessing URLs under `/1.0/containers/<name>/metadata`. It can be used to edit a container before publishing an image from it.

`container_snapshot_stateful_migration`

This enables migrating stateful container snapshots to new containers.

`storage_driver_ceph`

This adds a Ceph storage driver.

`storage_ceph_user_name`

This adds the ability to specify the Ceph user.

`instance_types`

This adds the `instance_type` field to the container creation request. Its value is expanded to LXD resource limits.

`storage_volatile_initial_source`

This records the actual source passed to LXD during storage pool creation.

`storage_ceph_force_osd_reuse`

This introduces the `ceph.osd.force_reuse` property for the Ceph storage driver. When set to `true` LXD will reuse an OSD storage pool that is already in use by another LXD instance.

`storage_block_filesystem_btrfs`

This adds support for Btrfs as a storage volume file system, in addition to `ext4` and `xfs`.

`resources`

This adds support for querying a LXD daemon for the system resources it has available.

`kernel_limits`

This adds support for setting process limits such as maximum number of open files for the container via `nofile`. The format is `limits.kernel.[limit name]`.

`storage_api_volume_rename`

This adds support for renaming custom storage volumes.

`network_sriov`

This adds support for SR-IOV enabled network devices.

`console`

This adds support to interact with the container console device and console log.

restrict_devlxd

A new *security.devlxd* (page 435) container configuration key was introduced. The key controls whether the /dev/lxd interface is made available to the instance. If set to false, this effectively prevents the container from interacting with the LXD daemon.

migration_pre_copy

This adds support for optimized memory transfer during live migration.

infiniband

This adds support to use InfiniBand network devices.

maas_network

This adds support for MAAS network integration.

When configured at the daemon level, it's then possible to attach a nic device to a particular MAAS subnet.

devlxd_events

This adds a WebSocket API to the devlxd socket.

When connecting to /1.0/events over the devlxd socket, you will now be getting a stream of events over WebSocket.

proxy

This adds a new proxy device type to containers, allowing forwarding of connections between the host and container.

network_dhcp_gateway

Introduces a new *ipv4.dhcp.gateway* (page 579) network configuration key to set an alternate gateway.

file_get_symlink

This makes it possible to retrieve symlinks using the file API.

network_leases

Adds a new /1.0/networks/NAME/leases API endpoint to query the lease database on bridges which run a LXD-managed DHCP server.

unix_device_hotplug

This adds support for the *required* (page 504) property for Unix devices.

`storage_api_local_volume_handling`

This add the ability to copy and move custom storage volumes locally in the same and between storage pools.

`operation_description`

Adds a `description` field to all operations.

`clustering`

Clustering API for LXD.

This includes the following new endpoints (see *RESTful API* (page 618) for details):

- `GET /1.0/cluster`
- `UPDATE /1.0/cluster`
- `GET /1.0/cluster/members`
- `GET /1.0/cluster/members/<name>`
- `POST /1.0/cluster/members/<name>`
- `DELETE /1.0/cluster/members/<name>`

The following existing endpoints have been modified:

- `POST /1.0/containers` accepts a new `target` query parameter
- `POST /1.0/storage-pools` accepts a new `target` query parameter
- `GET /1.0/storage-pool/<name>` accepts a new `target` query parameter
- `POST /1.0/storage-pool/<pool>/volumes/<type>` accepts a new `target` query parameter
- `GET /1.0/storage-pool/<pool>/volumes/<type>/<name>` accepts a new `target` query parameter
- `POST /1.0/storage-pool/<pool>/volumes/<type>/<name>` accepts a new `target` query parameter
- `PUT /1.0/storage-pool/<pool>/volumes/<type>/<name>` accepts a new `target` query parameter
- `PATCH /1.0/storage-pool/<pool>/volumes/<type>/<name>` accepts a new `target` query parameter
- `DELETE /1.0/storage-pool/<pool>/volumes/<type>/<name>` accepts a new `target` query parameter
- `POST /1.0/networks` accepts a new `target` query parameter
- `GET /1.0/networks/<name>` accepts a new `target` query parameter

event_lifecycle

This adds a new `lifecycle` message type to the events API.

storage_api_remote_volume_handling

This adds the ability to copy and move custom storage volumes between remote.

nvidia_runtime

Adds a *nvidia.runtime* (page 430) configuration option for containers, setting this to `true` will have the NVIDIA runtime and CUDA libraries passed to the container.

container_mount_propagation

This adds a new *propagation* (page 483) option to the disk device type, allowing the configuration of kernel mount propagation.

container_backup

Add container backup support.

This includes the following new endpoints (see *RESTful API* (page 618) for details):

- `GET /1.0/containers/<name>/backups`
- `POST /1.0/containers/<name>/backups`
- `GET /1.0/containers/<name>/backups/<name>`
- `POST /1.0/containers/<name>/backups/<name>`
- `DELETE /1.0/containers/<name>/backups/<name>`
- `GET /1.0/containers/<name>/backups/<name>/export`

The following existing endpoint has been modified:

- `POST /1.0/containers` accepts the new source type `backup`

devlxd_images

Adds a *security.devlxd.images* (page 435) configuration option for containers which controls the availability of a `/1.0/images/FINGERPRINT/export` API over `devlxd`. This can be used by a container running nested LXD to access raw images from the host.

container_local_cross_pool_handling

This enables copying or moving containers between storage pools on the same LXD instance.

proxy_unix

Add support for both Unix sockets and abstract Unix sockets in proxy devices. They can be used by specifying the address as `unix:/path/to/unix.sock` (normal socket) or `unix:@/tmp/unix.sock` (abstract socket).

Supported connections are now:

- `TCP <-> TCP`
- `UNIX <-> UNIX`
- `TCP <-> UNIX`
- `UNIX <-> TCP`

`proxy_udp`

Add support for UDP in proxy devices.

Supported connections are now:

- `TCP <-> TCP`
- `UNIX <-> UNIX`
- `TCP <-> UNIX`
- `UNIX <-> TCP`
- `UDP <-> UDP`
- `TCP <-> UDP`
- `UNIX <-> UDP`

`clustering_join`

This makes `GET /1.0/cluster` return information about which storage pools and networks are required to be created by joining nodes and which node-specific configuration keys they are required to use when creating them. Likewise the `PUT /1.0/cluster` endpoint now accepts the same format to pass information about storage pools and networks to be automatically created before attempting to join a cluster.

`proxy_tcp_udp_multi_port_handling`

Adds support for forwarding traffic for multiple ports. Forwarding is allowed between a range of ports if the port range is equal for source and target (for example `1.2.3.4 0-1000 -> 5.6.7.8 1000-2000`) and between a range of source ports and a single target port (for example `1.2.3.4 0-1000 -> 5.6.7.8 1000`).

`network_state`

Adds support for retrieving a network's state.

This adds the following new endpoint (see *RESTful API* (page 618) for details):

- `GET /1.0/networks/<name>/state`

`proxy_unix_dac_properties`

This adds support for GID, UID, and mode properties for non-abstract Unix sockets.

container_protection_delete

Enables setting the *security.protection.delete* (page 436) field which prevents containers from being deleted if set to `true`. Snapshots are not affected by this setting.

proxy_priv_drop

Adds *security.uid* (page 502) and *security.gid* (page 502) for the proxy devices, allowing privilege dropping and effectively changing the UID/GID used for connections to Unix sockets too.

pprof_http

This adds a new *core.debug_address* (page 402) configuration option to start a debugging HTTP server.

That server currently includes a `pprof` API and replaces the old `cpu-profile`, `memory-profile` and `print-goroutines` debug options.

proxy_haproxy_protocol

Adds a *proxy_protocol* (page 502) key to the proxy device which controls the use of the HAProxy PROXY protocol header.

network_hwaddr

Adds a *bridge.hwaddr* (page 576) key to control the MAC address of the bridge.

proxy_nat

This adds optimized UDP/TCP proxying. If the configuration allows, proxying will be done via `iptables` instead of proxy devices.

network_nat_order

This introduces the *ipv4.nat.order* (page 580) and *ipv6.nat.order* (page 582) configuration keys for LXD bridges. Those keys control whether to put the LXD rules before or after any pre-existing rules in the chain.

container_full

This introduces a new `recursion=2` mode for `GET /1.0/containers` which allows for the retrieval of all container structs, including the state, snapshots and backup structs.

This effectively allows for *lxc list* (page 785) to get all it needs in one query.

backup_compression

This introduces a new *backups.compression_algorithm* (page 411) configuration key which allows configuration of backup compression.

nvidia_runtime_config

This introduces a few extra configuration keys when using *nvidia.runtime* (page 430) and the `libnvidia-container` library. Those keys translate pretty much directly to the matching NVIDIA container environment variables:

- *nvidia.driver.capabilities* (page 429) => `NVIDIA_DRIVER_CAPABILITIES`
- *nvidia.require.cuda* (page 430) => `NVIDIA_REQUIRE_CUDA`
- *nvidia.require.driver* (page 430) => `NVIDIA_REQUIRE_DRIVER`

storage_api_volume_snapshots

Add support for storage volume snapshots. They work like container snapshots, only for volumes.

This adds the following new endpoint (see *RESTful API* (page 618) for details):

- `GET /1.0/storage-pools/<pool>/volumes/<type>/<name>/snapshots`
- `POST /1.0/storage-pools/<pool>/volumes/<type>/<name>/snapshots`
- `GET /1.0/storage-pools/<pool>/volumes/<type>/<volume>/snapshots/<name>`
- `PUT /1.0/storage-pools/<pool>/volumes/<type>/<volume>/snapshots/<name>`
- `POST /1.0/storage-pools/<pool>/volumes/<type>/<volume>/snapshots/<name>`
- `DELETE /1.0/storage-pools/<pool>/volumes/<type>/<volume>/snapshots/<name>`

storage_unmapped

Introduces a new `security.unmapped` Boolean on storage volumes.

Setting it to `true` will flush the current map on the volume and prevent any further idmap tracking and remapping on the volume.

This can be used to share data between isolated containers after attaching it to the container which requires write access.

projects

Add a new project API, supporting creation, update and deletion of projects.

Projects can hold containers, profiles or images at this point and let you get a separate view of your LXD resources by switching to it.

network_vxlan_ttl

This adds a new *tunnel.NAME.ttl* (page 586) network configuration option which makes it possible to raise the TTL on VXLAN tunnels.

container_incremental_copy

This adds support for incremental container copy. When copying a container using the `--refresh` flag, only the missing or outdated files will be copied over. Should the target container not exist yet, a normal copy operation is performed.

usb_optional_vendorid

As the name implies, the *vendorid* (page 491) field on USB devices attached to containers has now been made optional, allowing for all USB devices to be passed to a container (similar to what's done for GPUs).

snapshot_scheduling

This adds support for snapshot scheduling. It introduces three new configuration keys: `snapshots.schedule`, `snapshots.schedule.stopped`, and `snapshots.pattern`. Snapshots can be created automatically up to every minute.

snapshots_schedule_aliases

Snapshot schedule can be configured by a comma-separated list of schedule aliases. Available aliases are `<@hourly>` `<@daily>` `<@midnight>` `<@weekly>` `<@monthly>` `<@annually>` `<@yearly>` `<@startup>` for instances, and `<@hourly>` `<@daily>` `<@midnight>` `<@weekly>` `<@monthly>` `<@annually>` `<@yearly>` for storage volumes.

container_copy_project

Introduces a `project` field to the container source JSON object, allowing for copy/move of containers between projects.

clustering_server_address

This adds support for configuring a server network address which differs from the REST API client network address. When bootstrapping a new cluster, clients can set the new *cluster.https_address* (page 407) configuration key to specify the address of the initial server. When joining a new server, clients can set the *core.https_address* (page 402) configuration key of the joining server to the REST API address the joining server should listen at, and set the `server_address` key in the `PUT /1.0/cluster` API to the address the joining server should use for clustering traffic (the value of `server_address` will be automatically copied to the `cluster.https_address` configuration key of the joining server).

clustering_image_replication

Enable image replication across the nodes in the cluster. A new *cluster.images_minimal_replica* (page 407) configuration key was introduced can be used to specify to the minimal numbers of nodes for image replication.

container_protection_shift

Enables setting the *security.protection.shift* (page 437) option which prevents containers from having their file system shifted.

snapshot_expiry

This adds support for snapshot expiration. The task is run minutely. The configuration option *snapshots.expiry* (page 441) takes an expression in the form of `1M 2H 3d 4w 5m 6y` (1 minute, 2 hours, 3 days, 4 weeks, 5 months, 6 years), however not all parts have to be used.

Snapshots which are then created will be given an expiry date based on the expression. This expiry date, defined by `expires_at`, can be manually edited using the API or *lxc config edit* (page 744). Snapshots with a valid expiry date will be removed when the task in run. Expiry can be disabled by setting `expires_at` to an empty string or `0001-01-01T00:00:00Z` (zero time). This is the default if `snapshots.expiry` is not set.

This adds the following new endpoint (see *RESTful API* (page 618) for details):

- `PUT /1.0/containers/<name>/snapshots/<name>`

snapshot_expiry_creation

Adds `expires_at` to container creation, allowing for override of a snapshot's expiry at creation time.

network_leases_location

Introduces a `Location` field in the leases list. This is used when querying a cluster to show what node a particular lease was found on.

resources_cpu_socket

Add Socket field to CPU resources in case we get out of order socket information.

resources_gpu

Add a new GPU struct to the server resources, listing all usable GPUs on the system.

resources_numa

Shows the NUMA node for all CPUs and GPUs.

kernel_features

Exposes the state of optional kernel features through the server environment.

id_map_current

This introduces a new internal *volatile.idmap.current* (page 446) key which is used to track the current mapping for the container.

This effectively gives us:

- *volatile.last_state.idmap* (page 446) => On-disk idmap
- *volatile.idmap.current* (page 446) => Current kernel map
- *volatile.idmap.next* (page 446) => Next on-disk idmap

This is required to implement environments where the on-disk map isn't changed but the kernel map is (e.g. `idmapped mounts`).

event_location

Expose the location of the generation of API events.

storage_api_remote_volume_snapshots

This allows migrating storage volumes including their snapshots.

network_nat_address

This introduces the *ipv4.nat.address* (page 580) and *ipv6.nat.address* (page 582) configuration keys for LXD bridges. Those keys control the source address used for outbound traffic from the bridge.

container_nic_routes

This introduces the *ipv4.routes* (page 451) and *ipv6.routes* (page 451) properties on `nic` type devices. This allows adding static routes on host to container's NIC.

cluster_internal_copy

This makes it possible to do a normal `POST /1.0/containers` to copy a container between cluster nodes with LXD internally detecting whether a migration is required.

seccomp_notify

If the kernel supports `seccomp`-based syscall interception LXD can be notified by a container that a registered syscall has been performed. LXD can then decide to trigger various actions.

lxc_features

This introduces the `lxc_features` section output from the *lxc info* (page 782) command via the `GET /1.0` route. It outputs the result of checks for key features being present in the underlying LXC library.

container_nic_ipvlan

This introduces the `ipvlan` `nic` device type.

network_vlan_sriov

This introduces VLAN (*vlan* (page 460)) and MAC filtering (*security.mac_filtering* (page 459)) support for SR-IOV devices.

`storage_cephfs`

Add support for CephFS as a storage pool driver. This can only be used for custom volumes, images and containers should be on Ceph (RBD) instead.

`container_nic_ipfilter`

This introduces container IP filtering (*security.ipv4_filtering* (page 454) and *security.ipv6_filtering* (page 454)) support for `bridged` NIC devices.

`resources_v2`

Rework the resources API at `/1.0/resources`, especially:

- CPU
  - Fix reporting to track sockets, cores and threads
  - Track NUMA node per core
  - Track base and turbo frequency per socket
  - Track current frequency per core
  - Add CPU cache information
  - Export the CPU architecture
  - Show online/offline status of threads
- Memory
  - Add huge-pages tracking
  - Track memory consumption per NUMA node too
- GPU
  - Split DRM information to separate struct
  - Export device names and nodes in DRM struct
  - Export device name and node in NVIDIA struct
  - Add SR-IOV VF tracking

`container_exec_user_group_cwd`

Adds support for specifying `User`, `Group` and `Cwd` during `POST /1.0/containers/NAME/exec`.

`container_syscall_intercept`

Adds the `security.syscalls.intercept.*` configuration keys to control what system calls will be intercepted by LXD and processed with elevated permissions.

`container_disk_shift`

Adds the *`shift`* (page 484) property on `disk` devices which controls the use of the `idmapped mounts` overlay.

`storage_shifted`

Introduces a new `security.shifted` Boolean on storage volumes.

Setting it to `true` will allow multiple isolated containers to attach the same storage volume while keeping the file system writable from all of them.

This makes use of `idmapped mounts` as an overlay file system.

`resources_infiniband`

Export InfiniBand character device information (`issm`, `umad`, `uverb`) as part of the resources API.

`daemon_storage`

This introduces two new configuration keys *`storage.images_volume`* (page 413) and *`storage.backups_volume`* (page 413) to allow for a storage volume on an existing pool be used for storing the daemon-wide images and backups artifacts.

`instances`

This introduces the concept of instances, of which currently the only type is `container`.

`image_types`

This introduces support for a new Type field on images, indicating what type of images they are.

`resources_disk_sata`

Extends the disk resource API struct to include:

- Proper detection of SATA devices (type)
- Device path
- Drive RPM
- Block size
- Firmware version
- Serial number

`clustering_roles`

This adds a new `roles` attribute to cluster entries, exposing a list of roles that the member serves in the cluster.

images_expiry

This allows for editing of the expiry date on images.

resources_network_firmware

Adds a `FirmwareVersion` field to network card entries.

backup_compression_algorithm

This adds support for a `compression_algorithm` property when creating a backup (`POST /1.0/containers/<name>/backups`).

Setting this property overrides the server default value (*backups.compression_algorithm* (page 411)).

ceph_data_pool_name

This adds support for an optional argument (*ceph.osd.data_pool_name* (page 536)) when creating storage pools using Ceph RBD, when this argument is used the pool will store it's actual data in the pool specified with `data_pool_name` while keeping the metadata in the pool specified by `pool_name`.

container_syscall_intercept_mount

Adds the *security.syscalls.intercept.mount* (page 440), *security.syscalls.intercept.mount.allowed* (page 440), and *security.syscalls.intercept.mount.shift* (page 440) configuration keys to control whether and how the `mount` system call will be intercepted by LXD and processed with elevated permissions.

compression_squashfs

Adds support for importing/exporting of images/backups using SquashFS file system format.

container_raw_mount

This adds support for passing in raw mount options for disk devices.

container_nic_routed

This introduces the `routed` `nic` device type.

container_syscall_intercept_mount_fuse

Adds the *security.syscalls.intercept.mount.fuse* (page 440) key. It can be used to redirect file-system mounts to their fuse implementation. To this end, set e.g. `security.syscalls.intercept.mount.fuse=ext4=fuse2fs`.

container_disk_ceph

This allows for existing a Ceph RBD or CephFS to be directly connected to a LXD container.

virtual-machines

Add virtual machine support.

image_profiles

Allows a list of profiles to be applied to an image when launching a new container.

clustering_architecture

This adds a new `architecture` attribute to cluster members which indicates a cluster member's architecture.

resources_disk_id

Add a new `device_id` field in the disk entries on the resources API.

storage_lvm_stripes

This adds the ability to use LVM stripes on normal volumes and thin pool volumes.

vm_boot_priority

Adds a `boot.priority` property on NIC and disk devices to control the boot order.

unix_hotplug_devices

Adds support for Unix char and block device hotplugging.

api_filtering

Adds support for filtering the result of a GET request for instances and images.

instance_nic_network

Adds support for the `network` property on a NIC device to allow a NIC to be linked to a managed network. This allows it to inherit some of the network's settings and allows better validation of IP settings.

clustering_sizing

Support specifying a custom values for database voters and standbys. The new *`cluster.max_voters`* (page 408) and *`cluster.max_standby`* (page 408) configuration keys were introduced to specify to the ideal number of database voter and standbys.

firewall_driver

Adds the Firewall property to the ServerEnvironment struct indicating the firewall driver being used.

storage_lvm_vg_force_reuse

Introduces the ability to create a storage pool from an existing non-empty volume group. This option should be used with care, as LXD can then not guarantee that volume name conflicts won't occur with non-LXD created volumes in the same volume group. This could also potentially lead to LXD deleting a non-LXD volume should name conflicts occur.

container_syscall_intercept_hugetlbfs

When mount syscall interception is enabled and hugetlbfs is specified as an allowed file system type LXD will mount a separate hugetlbfs instance for the container with the UID and GID mount options set to the container's root UID and GID. This ensures that processes in the container can use huge pages.

limits_hugepages

This allows to limit the number of huge pages a container can use through the hugetlb cgroup. This means the hugetlb cgroup needs to be available. Note, that limiting huge pages is recommended when intercepting the mount syscall for the hugetlbfs file system to avoid allowing the container to exhaust the host's huge pages resources.

container_nic_routed_gateway

This introduces the *ipv4.gateway* (page 474) and *ipv6.gateway* (page 475) NIC configuration keys that can take a value of either auto or none. The default value for the key if unspecified is auto. This will cause the current behavior of a default gateway being added inside the container and the same gateway address being added to the host-side interface. If the value is set to none then no default gateway nor will the address be added to the host-side interface. This allows multiple routed NIC devices to be added to a container.

projects_restrictions

This introduces support for the *restricted* (page 514) configuration key on project, which can prevent the use of security-sensitive features in a project.

custom_volume_snapshot_expiry

This allows custom volume snapshots to expiry. Expiry dates can be set individually, or by setting the snapshots.expiry configuration key on the parent custom volume which then automatically applies to all created snapshots.

volume_snapshot_scheduling

This adds support for custom volume snapshot scheduling. It introduces two new configuration keys: snapshots.schedule and snapshots.pattern. Snapshots can be created automatically up to every minute.

`trust_ca_certificates`

This allows for checking client certificates trusted by the provided CA (`server.ca`). It can be enabled by setting *`core.trust_ca_certificates`* (page 405) to `true`. If enabled, it will perform the check, and bypass the trusted password if `true`. An exception will be made if the connecting client certificate is in the provided CRL (`ca.crl`). In this case, it will ask for the password.

`snapshot_disk_usage`

This adds a new `size` field to the output of `/1.0/instances/<name>/snapshots/<snapshot>` which represents the disk usage of the snapshot.

`clustering_edit_roles`

This adds a writable endpoint for cluster members, allowing the editing of their roles.

`container_nic_routed_host_address`

This introduces the *`ipv4.host_address`* (page 474) and *`ipv6.host_address`* (page 475) NIC configuration keys that can be used to control the host-side `veth` interface's IP addresses. This can be useful when using multiple routed NICs at the same time and needing a predictable next-hop address to use.

This also alters the behavior of *`ipv4.gateway`* (page 474) and *`ipv6.gateway`* (page 475) NIC configuration keys. When they are set to `auto` the container will have its default gateway set to the value of `ipv4.host_address` or `ipv6.host_address` respectively.

The default values are:

`ipv4.host_address: 169.254.0.1` `ipv6.host_address: fe80::1`

This is backward compatible with the previous default behavior.

`container_nic_ipvlan_gateway`

This introduces the *`ipv4.gateway`* (page 468) and *`ipv6.gateway`* (page 469) NIC configuration keys that can take a value of either `auto` or `none`. The default value for the key if unspecified is `auto`. This will cause the current behavior of a default gateway being added inside the container and the same gateway address being added to the host-side interface. If the value is set to `none` then no default gateway nor will the address be added to the host-side interface. This allows multiple IPVLAN NIC devices to be added to a container.

`resources_usb_pci`

This adds USB and PCI devices to the output of `/1.0/resources`.

`resources_cpu_threads_numa`

This indicates that the `numa_node` field is now recorded per-thread rather than per core as some hardware apparently puts threads in different NUMA domains.

resources_cpu_core_die

Exposes the die_id information on each core.

api_os

This introduces two new fields in /1.0, os and os_version.

Those are taken from the OS-release data on the system.

container_nic_routed_host_table

This introduces the *ipv4.host_table* (page 475) and *ipv6.host_table* (page 476) NIC configuration keys that can be used to add static routes for the instance's IPs to a custom policy routing table by ID.

container_nic_ipvlan_host_table

This introduces the *ipv4.host_table* (page 468) and *ipv6.host_table* (page 469) NIC configuration keys that can be used to add static routes for the instance's IPs to a custom policy routing table by ID.

container_nic_ipvlan_mode

This introduces the *mode* (page 469) NIC configuration key that can be used to switch the ipvlan mode into either l2 or l3s. If not specified, the default value is l3s (which is the old behavior).

In l2 mode the *ipv4.address* (page 468) and *ipv6.address* (page 468) keys will accept addresses in either CIDR or singular formats. If singular format is used, the default subnet size is taken to be /24 and /64 for IPv4 and IPv6 respectively.

In l2 mode the *ipv4.gateway* (page 468) and *ipv6.gateway* (page 469) keys accept only a singular IP address.

resources_system

This adds system information to the output of /1.0/resources.

images_push_relay

This adds the push and relay modes to image copy. It also introduces the following new endpoint:

- POST 1.0/images/<fingerprint>/export

network_dns_search

This introduces the dns.search configuration option on networks.

`container_nic_routed_limits`

This introduces *limits.ingress* (page 476), *limits.egress* (page 476) and *limits.max* (page 476) for routed NICs.

`instance_nic_bridged_vlan`

This introduces the *vlan* (page 455) and *vlan.tagged* (page 455) settings for `bridged` NICs.

`vlan` specifies the non-tagged VLAN to join, and `vlan.tagged` is a comma-delimited list of tagged VLANs to join.

`network_state_bond_bridge`

This adds a `bridge` and `bond` section to the `/1.0/networks/NAME/state` API.

Those contain additional state information relevant to those particular types.

Bond:

- Mode
- Transmit hash
- Up delay
- Down delay
- MII frequency
- MII state
- Lower devices

Bridge:

- ID
- Forward delay
- STP mode
- Default VLAN
- VLAN filtering
- Upper devices

`resources_cpu_isolated`

Add an `Isolated` property on CPU threads to indicate if the thread is physically `Online` but is configured not to accept tasks.

`usedby_consistency`

This extension indicates that `UsedBy` should now be consistent with suitable `?project=` and `?target=` when appropriate.

The 5 entities that have `UsedBy` are:

- Profiles

- Projects
- Networks
- Storage pools
- Storage volumes

custom_block_volumes

This adds support for creating and attaching custom block volumes to instances. It introduces the new `--type` flag when creating custom storage volumes, and accepts the values `fs` and `block`.

clustering_failure_domains

This extension adds a new `failure_domain` field to the `PUT /1.0/cluster/<node>` API, which can be used to set the failure domain of a node.

container_syscall_filtering_allow_deny_syntax

A number of new syscalls related container configuration keys were updated.

- *security.syscalls.deny_default* (page 439)
- *security.syscalls.deny_compat* (page 439)
- *security.syscalls.deny* (page 438)
- *security.syscalls.allow* (page 438)

Support for the offensively named variants was removed.

resources_gpu_mdev

Expose available mediated device profiles and devices in `/1.0/resources`.

console_vga_type

This extends the `/1.0/console` endpoint to take a `?type=` argument, which can be set to `console` (default) or `vga` (the new type added by this extension).

When doing a `POST` to `/1.0/<instance name>/console?type=vga` the data WebSocket returned by the operation in the metadata field will be a bidirectional proxy attached to a SPICE Unix socket of the target virtual machine.

projects_limits_disk

Add *limits.disk* (page 512) to the available project configuration keys. If set, it limits the total amount of disk space that instances volumes, custom volumes and images volumes can use in the project.

`network_type_macvlan`

Adds support for additional network type `macvlan` and adds *`parent`* (page 595) configuration key for this network type to specify which parent interface should be used for creating NIC device interfaces on top of.

Also adds *`network`* (page 457) configuration key support for `macvlan` NICs to allow them to specify the associated network of the same type that they should use as the basis for the NIC device.

`network_type_sriov`

Adds support for additional network type `sriov` and adds *`parent`* (page 601) configuration key for this network type to specify which parent interface should be used for creating NIC device interfaces on top of.

Also adds *`network`* (page 459) configuration key support for `sriov` NICs to allow them to specify the associated network of the same type that they should use as the basis for the NIC device.

`container_syscall_intercept_bpf_devices`

This adds support to intercept the `bpf` syscall in containers. Specifically, it allows to manage device cgroup `bpf` programs.

`network_type_ovn`

Adds support for additional network type `ovn` with the ability to specify a `bridge` type network as the `parent`.

Introduces a new NIC device type of `ovn` which allows the `network` configuration key to specify which `ovn` type network they should connect to.

Also introduces two new global configuration keys that apply to all `ovn` networks and NIC devices:

- *`network.ovn.integration_bridge`* (page 413) - the OVS integration bridge to use.
- *`network.ovn.northbound_connection`* (page 413) - the OVN northbound database connection string.

`projects_networks`

Adds the *`features.networks`* (page 510) configuration key to projects and the ability for a project to hold networks.

`projects_networks_restricted_uplinks`

Adds the *`restricted.networks.uplinks`* (page 519) project configuration key to indicate (as a comma-delimited list) which networks the networks created inside the project can use as their uplink network.

custom_volume_backup

Add custom volume backup support.

This includes the following new endpoints (see *RESTful API* (page 618) for details):

- GET /1.0/storage-pools/<pool>/<type>/<volume>/backups
- POST /1.0/storage-pools/<pool>/<type>/<volume>/backups
- GET /1.0/storage-pools/<pool>/<type>/<volume>/backups/<name>
- POST /1.0/storage-pools/<pool>/<type>/<volume>/backups/<name>
- DELETE /1.0/storage-pools/<pool>/<type>/<volume>/backups/<name>
- GET /1.0/storage-pools/<pool>/<type>/<volume>/backups/<name>/export

The following existing endpoint has been modified:

- POST /1.0/storage-pools/<pool>/<type>/<volume> accepts the new source type backup

backup_override_name

Adds Name field to InstanceBackupArgs to allow specifying a different instance name when restoring a backup.

Adds Name and PoolName fields to StoragePoolVolumeBackupArgs to allow specifying a different volume name when restoring a custom volume backup.

storage_rsync_compression

Adds rsync.compression configuration key to storage pools. This key can be used to disable compression in rsync while migrating storage pools.

network_type_physical

Adds support for additional network type physical that can be used as an uplink for ovn networks.

The interface specified by *parent* (page 599) on the physical network will be connected to the ovn network's gateway.

network_ovn_external_subnets

Adds support for ovn networks to use external subnets from uplink networks.

Introduces the *ipv4.routes* (page 597) and *ipv6.routes* (page 598) setting on physical networks that defines the external routes allowed to be used in child OVN networks in their *ipv4.routes.external* (page 464) and *ipv6.routes.external* (page 465) settings.

Introduces the *restricted.networks.subnets* (page 519) project setting that specifies which external subnets are allowed to be used by OVN networks inside the project (if not set then all routes defined on the uplink network are allowed).

network_ovn_nat

Adds support for *ipv4.nat* (page 590) and *ipv6.nat* (page 591) settings on ovn networks.

When creating the network if these settings are unspecified, and an equivalent IP address is being generated for the subnet, then the appropriate NAT setting will added set to true.

If the setting is missing then the value is taken as false.

network_ovn_external_routes_remove

Removes the settings ipv4.routes.external and ipv6.routes.external from ovn networks.

The equivalent settings on the ovn NIC type can be used instead for this, rather than having to specify them both at the network and NIC level.

tpm_device_type

This introduces the tpm device type.

storage_zfs_clone_copy_rebase

This introduces rebase as a value for *zfs.clone_copy* (page 566) causing LXD to track down any image dataset in the ancestry line and then perform send/receive on top of that.

gpu_mdev

This adds support for virtual GPUs (vGPUs). It introduces the *mdev* (page 494) configuration key for GPU devices which takes a supported mdev type, e.g. i915-GVTg_V5_4.

resources_pci_iommu

This adds the IOMMUGroup field for PCI entries in the resources API.

resources_network_usb

Adds the usb_address field to the network card entries in the resources API.

resources_disk_address

Adds the usb_address and pci_address fields to the disk entries in the resources API.

network_physical_ovn_ingress_mode

Adds *ovn.ingress_mode* (page 599) setting for physical networks.

Sets the method that OVN NIC external IPs will be advertised on uplink network.

Either l2proxy (proxy ARP/NDP) or routed.

network_ovn_dhcp

Adds *ipv4.dhcp* (page 590) and *ipv6.dhcp* (page 591) settings for `ovn` networks.

Allows DHCP (and RA for IPv6) to be disabled. Defaults to on.

network_physical_routes_anycast

Adds *ipv4.routes.anycast* (page 598) and *ipv6.routes.anycast* (page 598) Boolean settings for `physical` networks. Defaults to `false`.

Allows OVN networks using physical network as uplink to relax external subnet/route overlap detection when used with *ovn.ingress_mode* (page 599) set to `routed`.

projects_limits_instances

Adds *limits.instances* (page 512) to the available project configuration keys. If set, it limits the total number of instances (VMs and containers) that can be used in the project.

network_state_vlan

This adds a `vlan` section to the `/1.0/networks/NAME/state` API.

Those contain additional state information relevant to VLAN interfaces:

- `lower_device`
- `vid`

instance_nic_bridged_port_isolation

This adds the *security.port_isolation* (page 454) field for bridged NIC instances.

instance_bulk_state_change

Adds the following endpoint for bulk state change (see *RESTful API* (page 618) for details):

- `PUT /1.0/instances`

network_gvrp

This adds an optional `gvrp` property to `macvlan` and `physical` networks, and to `ipvlan`, `macvlan`, `routed` and `physical` NIC devices.

When set, this specifies whether the VLAN should be registered using GARP VLAN Registration Protocol. Defaults to `false`.

instance_pool_move

This adds a `pool` field to the `POST /1.0/instances/NAME` API, allowing for easy move of an instance root disk between pools.

gpu_sriov

This adds support for SR-IOV enabled GPUs. It introduces the `sriov` GPU type property.

pci_device_type

This introduces the `pci` device type.

storage_volume_state

Add new `/1.0/storage-pools/POOL/volumes/VOLUME/state` API endpoint to get usage data on a volume.

network_acl

This adds the concept of network ACLs to API under the API endpoint prefix `/1.0/network-acls`.

migration_stateful

Add a new *migration.stateful* (page 429) configuration key.

disk_state_quota

This introduces the *size.state* (page 484) device configuration key on `disk` devices.

storage_ceph_features

Adds a new *ceph.rbd.features* (page 537) configuration key on storage pools to control the RBD features used for new volumes.

projects_compression

Adds new *backups.compression_algorithm* (page 520) and *images.compression_algorithm* (page 520) configuration keys which allows configuration of backup and image compression per-project.

projects_images_remote_cache_expiry

Add new *images.remote_cache_expiry* (page 409) configuration key to projects, allowing for set number of days after which an unused cached remote image will be flushed.

certificate_project

Adds a new `restricted` property to certificates in the API as well as `projects` holding a list of project names that the certificate has access to.

network_ovn_acl

Adds a new `security.acls` property to OVN networks and OVN NICs, allowing Network ACLs to be applied.

projects_images_auto_update

Adds new *images.auto_update_cached* (page 409) and *images.auto_update_interval* (page 409) configuration keys which allows configuration of images auto update in projects

projects_restricted_cluster_target

Adds new *restricted.cluster.target* (page 514) configuration key to project which prevent the user from using –target to specify what cluster member to place a workload on or the ability to move a workload between members.

images_default_architecture

Adds new *images.default_architecture* (page 409) global configuration key and matching per-project key which lets user tell LXD what architecture to go with when no specific one is specified as part of the image request.

network_ovn_acl_defaults

Adds new `security.acls.default.{in,e}gress.action` and `security.acls.default.{in,e}gress.logged` configuration keys for OVN networks and NICs. This replaces the removed ACL `default.action` and `default.logged` keys.

gpu_mig

This adds support for NVIDIA MIG. It introduces the `mig` GPU type and associated configuration keys.

project_usage

Adds an API endpoint to get current resource allocations in a project. Accessible at API `GET /1.0/projects/<name>/state`.

network_bridge_acl

Adds a new *security.acls* (page 584) configuration key to `bridge` networks, allowing Network ACLs to be applied.

Also adds `security.acls.default.{in,e}gress.action` and `security.acls.default.{in,e}gress.logged` configuration keys for specifying the default behavior for unmatched traffic.

warnings

Warning API for LXD.

This includes the following endpoints (see *Restful API* (page 618) for details):

- `GET /1.0/warnings`

- `GET /1.0/warnings/<uuid>`
- `PUT /1.0/warnings/<uuid>`
- `DELETE /1.0/warnings/<uuid>`

projects_restricted_backups_and_snapshots

Adds new *restricted.backups* (page 514) and *restricted.snapshots* (page 519) configuration keys to project which prevents the user from creation of backups and snapshots.

clustering_join_token

Adds `POST /1.0/cluster/members` API endpoint for requesting a join token used when adding new cluster members without using the trust password.

clustering_description

Adds an editable description to the cluster members.

server_trusted_proxy

This introduces support for *core.https_trusted_proxy* (page 403) which has LXD parse a HAProxy style connection header on such connections and if present, will rewrite the request's source address to that provided by the proxy server.

clustering_update_cert

Adds `PUT /1.0/cluster/certificate` endpoint for updating the cluster certificate across the whole cluster

storage_api_project

This adds support for copy/move custom storage volumes between projects.

server_instance_driver_operational

This modifies the `driver` output for the `/1.0` endpoint to only include drivers which are actually supported and operational on the server (as opposed to being included in LXD but not operational on the server).

server_supported_storage_drivers

This adds supported storage driver info to server environment info.

event_lifecycle_requestor_address

Adds a new address field to `lifecycle` requestor.

resources_gpu_usb

Add a new USBAddress (usb_address) field to ResourcesGPUCard (GPU entries) in the resources API.

clustering_evacuation

Adds POST /1.0/cluster/members/<name>/state endpoint for evacuating and restoring cluster members. It also adds the configuration keys *cluster.evacuate* (page 416) and *volatile.evacuate.origin* (page 445) for setting the evacuation method (auto, stop or migrate) and the origin of any migrated instance respectively.

network_ovn_nat_address

This introduces the *ipv4.nat.address* (page 590) and *ipv6.nat.address* (page 591) configuration keys for LXD ovn networks. Those keys control the source address used for outbound traffic from the OVN virtual network. These keys can only be specified when the OVN network's uplink network has *ovn.ingress_mode* (page 599) set to routed.

network_bgp

This introduces support for LXD acting as a BGP router to advertise routes to bridge and ovn networks.

This comes with the addition to global configuration of:

- *core.bgp_address* (page 401)
- *core.bgp_asn* (page 401)
- *core.bgp_routerid* (page 401)

The following network configurations keys (bridge and physical):

- bgp.peers.<name>.address
- bgp.peers.<name>.asn
- bgp.peers.<name>.password

The nexthop configuration keys (bridge):

- *bgp.ipv4.nexthop* (page 574)
- *bgp.ipv6.nexthop* (page 575)

And the following NIC-specific configuration keys (bridged NIC type):

- *ipv4.routes.external* (page 451)
- *ipv6.routes.external* (page 452)

network_forward

This introduces the networking address forward functionality. Allowing for bridge and ovn networks to define external IP addresses that can be forwarded to internal IP(s) inside their respective networks.

`custom_volume_refresh`

Adds support for refresh during volume migration.

`network_counters_errors_dropped`

This adds the received and sent errors as well as inbound and outbound dropped packets to the network counters.

`metrics`

This adds metrics to LXD. It returns metrics of running instances using the OpenMetrics format.

This includes the following endpoints:

- `GET /1.0/metrics`

`image_source_project`

Adds a new `project` field to `POST /1.0/images` allowing for the source project to be set at image copy time.

`clustering_config`

Adds new `config` property to cluster members with configurable key/value pairs.

`network_peer`

This adds network peering to allow traffic to flow between OVN networks without leaving the OVN subsystem.

`linux_sysctl`

Adds new `linux.sysctl.*` configuration keys allowing users to modify certain kernel parameters within containers.

`network_dns`

Introduces a built-in DNS server and zones API to provide DNS records for LXD instances.

This introduces the following server configuration key:

- *`core.dns_address`* (page 402)

The following network configuration key:

- `dns.zone.forward`
- `dns.zone.reverse.ipv4`
- `dns.zone.reverse.ipv6`

And the following project configuration key:

- *`restricted.networks.zones`* (page 519)

A new REST API is also introduced to manage DNS zones:

- `/1.0/network-zones` (GET, POST)
- `/1.0/network-zones/<name>` (GET, PUT, PATCH, DELETE)

`ovn_nic_acceleration`

Adds new *`acceleration`* (page 463) configuration key to OVN NICs which can be used for enabling hardware offloading. It takes the values `none` or `sriov`.

`certificate_self_renewal`

This adds support for renewing a client's own trust certificate.

`instance_project_move`

This adds a `project` field to the `POST /1.0/instances/NAME` API, allowing for easy move of an instance between projects.

`storage_volume_project_move`

This adds support for moving storage volume between projects.

`cloud_init`

This adds a new `cloud-init` configuration key namespace which contains the following keys:

- *`cloud-init.vendor-data`* (page 420)
- *`cloud-init.user-data`* (page 420)
- *`cloud-init.network-config`* (page 419)

It also adds a new endpoint `/1.0/devices` to `devlxd` which shows an instance's devices.

`network_dns_nat`

This introduces `network.nat` as a configuration option on network zones (DNS).

It defaults to the current behavior of generating records for all instances NICs but if set to `false`, it will instruct LXD to only generate records for externally reachable addresses.

`database_leader`

Adds new `database-leader` role which is assigned to cluster leader.

`instance_all_projects`

This adds support for displaying instances from all projects.

clustering_groups

Add support for grouping cluster members.

This introduces the following new endpoints:

- `/1.0/cluster/groups` (GET, POST)
- `/1.0/cluster/groups/<name>` (GET, POST, PUT, PATCH, DELETE)

The following project restriction is added:

- *restricted.cluster.groups* (page 514)

ceph_rbd_du

Adds a new *ceph.rbd.du* (page 537) Boolean on Ceph storage pools which allows disabling the use of the potentially slow `rbd du` calls.

instance_get_full

This introduces a new `recursion=1` mode for `GET /1.0/instances/{name}` which allows for the retrieval of all instance structs, including the state, snapshots and backup structs.

qemu_metrics

This adds a new *security.agent.metrics* (page 433) Boolean which defaults to `true`. When set to `false`, it doesn't connect to the `lxd-agent` for metrics and other state information, but relies on stats from QEMU.

gpu_mig_uuid

Adds support for the new MIG UUID format used by NVIDIA `470+` drivers (for example, `MIG-74c6a31a-fde5-5c61-973b-70e12346c202`), the `MIG-` prefix can be omitted

This extension supersedes old `mig.gi` and `mig.ci` parameters which are kept for compatibility with old drivers and cannot be set together.

event_project

Expose the project an API event belongs to.

clustering_evacuation_live

This adds `live-migrate` as a configuration option to *cluster.evacuate* (page 416), which forces live-migration of instances during cluster evacuation.

instance_allow_inconsistent_copy

Adds `allow_inconsistent` field to instance source on `POST /1.0/instances`. If `true`, rsync will ignore the `Partial transfer due to vanished source files` (code 24) error when creating an instance from a copy.

network_state_ovn

This adds an `ovn` section to the `/1.0/networks/NAME/state` API which contains additional state information relevant to OVN networks:

- chassis

storage_volume_api_filtering

Adds support for filtering the result of a GET request for storage volumes.

image_restrictions

This extension adds on to the image properties to include image restrictions/host requirements. These requirements help determine the compatibility between an instance and the host system.

storage_zfs_export

Introduces the ability to disable zpool export when unmounting pool by setting *zfs.export* (page 566).

network_dns_records

This extends the network zones (DNS) API to add the ability to create and manage custom records.

This adds:

- `GET /1.0/network-zones/ZONE/records`
- `POST /1.0/network-zones/ZONE/records`
- `GET /1.0/network-zones/ZONE/records/RECORD`
- `PUT /1.0/network-zones/ZONE/records/RECORD`
- `PATCH /1.0/network-zones/ZONE/records/RECORD`
- `DELETE /1.0/network-zones/ZONE/records/RECORD`

storage_zfs_reserve_space

Adds ability to set the `reservation`/`refreservation` ZFS property along with `quota`/`refquota`.

network_acl_log

Adds a new `GET /1.0/networks-acls/NAME/log` API to retrieve ACL firewall logs.

storage_zfs_blocksize

Introduces a new *zfs.blocksize* (page 569) property for ZFS storage volumes which allows to set volume block size.

`metrics_cpu_seconds`

This is used to detect whether LXD was fixed to output used CPU time in seconds rather than as milliseconds.

`instance_snapshot_never`

Adds a `@never` option to `snapshots.schedule` which allows disabling inheritance.

`certificate_token`

This adds token-based certificate addition to the trust store as a safer alternative to a trust password.

It adds the `token` field to `POST /1.0/certificates`.

`instance_nic_routed_neighbor_probe`

This adds the ability to disable the `routed` NIC IP neighbor probing for availability on the parent network.

Adds the *`ipv4.neighbor_probe`* (page 475) and *`ipv6.neighbor_probe`* (page 476) NIC settings. Defaulting to `true` if not specified.

`event_hub`

This adds support for `event-hub` cluster member role and the `ServerEventMode` environment field.

`agent_nic_config`

If set to `true`, on VM start-up the `lxd-agent` will apply NIC configuration to change the names and MTU of the instance NIC devices.

`projects_restricted_intercept`

Adds new *`restricted.containers.interception`* (page 515) configuration key to allow usually safe system call interception options.

`metrics_authentication`

Introduces a new *`core.metrics_authentication`* (page 403) server configuration option to allow for the `/1.0/metrics` endpoint to be generally available without client authentication.

`images_target_project`

Adds ability to copy image to a project different from the source.

cluster_migration_inconsistent_copy

Adds `allow_inconsistent` field to `POST /1.0/instances/<name>`. Set to `true` to allow inconsistent copying between cluster members.

cluster_ovn_chassis

Introduces a new `ovn-chassis` cluster role which allows for specifying what cluster member should act as an OVN chassis.

container_syscall_intercept_sched_setscheduler

Adds the *security.syscalls.intercept.sched_setscheduler* (page 441) to allow advanced process priority management in containers.

storage_lvm_thinpool_metadata_size

Introduces the ability to specify the thin pool metadata volume size via *lvm.thinpool_metadata_size* (page 558).

If this is not specified then the default is to let LVM pick an appropriate thin pool metadata volume size.

storage_volume_state_total

This adds `total` field to the `GET /1.0/storage-pools/{name}/volumes/{type}/{volume}/state` API.

instance_file_head

Implements HEAD on `/1.0/instances/NAME/file`.

instances_nic_host_name

This introduces the *instances.nic.host_name* (page 411) server configuration key that can take a value of either `random` or `mac`. The default value for the key if unspecified is `random`. If it is set to random then use the random host interface names. If it's set to `mac`, then generate a name in the form `lxd1122334455`.

image_copy_profile

Adds ability to modify the set of profiles when image is copied.

container_syscall_intercept_sysinfo

Adds the *security.syscalls.intercept.sysinfo* (page 441) to allow the `sysinfo` syscall to be populated with cgroup-based resource usage information.

clustering_evacuation_mode

This introduces a mode field to the evacuation request which allows for overriding the evacuation mode traditionally set through *cluster.evacuate* (page 416).

resources_pci_vpd

Adds a new VPD struct to the PCI resource entries. This struct extracts vendor provided data including the full product name and additional key/value configuration pairs.

qemu_raw_conf

Introduces a *raw.qemu.conf* (page 431) configuration key to override select sections of the generated qemu.conf.

storage_cephfs_fscache

Add support for fscache/cachefilesd on CephFS pools through a new *cephfs.fscache* (page 528) configuration option.

network_load_balancer

This introduces the networking load balancer functionality. Allowing ovn networks to define port(s) on external IP addresses that can be forwarded to one or more internal IP(s) inside their respective networks.

vsock_api

This introduces a bidirectional vsock interface which allows the lxd-agent and the LXD server to communicate better.

instance_ready_state

This introduces a new Ready state for instances which can be set using devlxd.

network_bgp_holdtime

This introduces a new bgp.peers.<name>.holdtime configuration key to control the BGP hold time for a particular peer.

storage_volumes_all_projects

This introduces the ability to list storage volumes from all projects.

metrics_memory_oom_total

This introduces a new lxd_memory_OOM_kills_total metric to the /1.0/metrics API. It reports the number of times the out of memory killer (OOM) has been triggered.

storage_buckets

This introduces the storage bucket API. It allows the management of S3 object storage buckets for storage pools.

storage_buckets_create_credentials

This updates the storage bucket API to return initial admin credentials at bucket creation time.

metrics_cpu_effective_total

This introduces a new `lxd_cpu_effective_total` metric to the `/1.0/metrics` API. It reports the total number of effective CPUs.

projects_networks_restricted_access

Adds the *restricted.networks.access* (page 518) project configuration key to indicate (as a comma-delimited list) which networks can be accessed inside the project. If not specified, all networks are accessible (assuming it is also allowed by the *restricted.devices.nic* (page 517) setting, described below).

This also introduces a change whereby network access is controlled by the project's *restricted.devices.nic* (page 517) setting:

- If `restricted.devices.nic` is set to `managed` (the default if not specified), only managed networks are accessible.
- If `restricted.devices.nic` is set to `allow`, all networks are accessible (dependent on the `restricted.networks.access` setting).
- If `restricted.devices.nic` is set to `block`, no networks are accessible.

storage_buckets_local

This introduces the ability to use storage buckets on local storage pools by setting the new *core.storage_buckets_address* (page 404) global configuration setting.

loki

This adds support for sending life cycle and logging events to a Loki server.

It adds the following global configuration keys:

- *loki.api.ca_cert* (page 410): CA certificate which can be used when sending events to the Loki server
- *loki.api.url* (page 410): URL to the Loki server (protocol, name or IP and port)
- *loki.auth.username* (page 410) and *loki.auth.password* (page 410): Used if Loki is behind a reverse proxy with basic authentication enabled
- *loki.labels* (page 410): Comma-separated list of values which are to be used as labels for Loki events.
- *loki.loglevel* (page 411): Minimum log level for events sent to the Loki server.

- *loki.types* (page 411): Types of events which are to be sent to the Loki server (`lifecycle` and/or `logging`).

acme

This adds ACME support, which allows Let's Encrypt[276] or other ACME services to issue certificates.

It adds the following global configuration keys:

- *acme.domain* (page 405): The domain for which the certificate should be issued.
- *acme.email* (page 405): The email address used for the account of the ACME service.
- *acme.ca_url* (page 405): The directory URL of the ACME service, defaults to `https://acme-v02.api.letsencrypt.org/directory`.

It also adds the following endpoint, which is required for the HTTP-01 challenge:

- `/.well-known/acme-challenge/<token>`

internal_metrics

This adds internal metrics to the list of metrics. These include:

- Total running operations
- Total active warnings
- Daemon uptime in seconds
- Go memory stats
- Number of goroutines

cluster_join_token_expiry

This adds an expiry to cluster join tokens which defaults to 3 hours, but can be changed by setting the *cluster.join_token_expiry* (page 408) configuration key.

remote_token_expiry

This adds an expiry to remote add join tokens. It can be set in the *core.remote_token_expiry* (page 404) configuration key, and default to no expiry.

storage_volumes_created_at

This change adds support for storing the creation date and time of storage volumes and their snapshots.

This adds the `CreatedAt` field to the `StorageVolume` and `StorageVolumeSnapshot` API types.

[276] https://letsencrypt.org/

cpu_hotplug

This adds CPU hotplugging for VMs. Hotplugging is disabled when using CPU pinning, because this would require hotplugging NUMA devices as well, which is not possible.

projects_networks_zones

This adds support for the *features.networks.zones* (page 510) project feature, which changes which project network zones are associated with when they are created. Previously network zones were tied to the value of *features.networks* (page 510), meaning they were created in the same project as networks were.

Now this has been decoupled from *features.networks* (page 510) to allow projects that share a network in the default project (i.e those with `features.networks=false`) to have their own project level DNS zones that give a project oriented "view" of the addresses on that shared network (which only includes addresses from instances in their project).

This also introduces a change to the network `dns.zone.forward` setting, which now accepts a comma-separated of DNS zone names (a maximum of one per project) in order to associate a shared network with multiple zones.

No change to the `dns.zone.reverse.*` settings have been made, they still only allow a single DNS zone to be set. However the resulting zone content that is generated now includes `PTR` records covering addresses from all projects that are referencing that network via one of their forward zones.

Existing projects that have `features.networks=true` will have `features.networks.zones=true` set automatically, but new projects will need to specify this explicitly.

instance_nic_txqueuelength

Adds a `txqueuelen` key to control the `txqueuelen` parameter of the NIC device.

cluster_member_state

Adds `GET /1.0/cluster/members/<member>/state` API endpoint and associated `ClusterMemberState` API response type.

instances_placement_scriptlet

Adds support for a Starlark scriptlet to be provided to LXD to allow customized logic that controls placement of new instances in a cluster.

The Starlark scriptlet is provided to LXD via the new global configuration option *instances.placement.scriptlet* (page 412).

storage_pool_source_wipe

Adds support for a `source.wipe` Boolean on the storage pool, indicating that LXD should wipe partition headers off the requested disk rather than potentially fail due to pre-existing file systems.

`zfs_block_mode`

This adds support for using ZFS block volumes allowing the use of different file systems on top of ZFS.

This adds the following new configuration options for ZFS storage pools:

- `volume.zfs.block_mode`
- `volume.block.mount_options`
- `volume.block.filesystem`

`instance_generation_id`

Adds support for instance generation ID. The VM or container generation ID will change whenever the instance's place in time moves backwards. As of now, the generation ID is only exposed through to VM type instances. This allows for the VM guest OS to reinitialize any state it needs to avoid duplicating potential state that has already occurred:

- *`volatile.uuid.generation`* (page 447)

`disk_io_cache`

This introduces a new *`io.cache`* (page 481) property to disk devices which can be used to override the VM caching behavior.

`amd_sev`

Adds support for AMD SEV (Secure Encrypted Virtualization) that can be used to encrypt the memory of a guest VM.

This adds the following new configuration options for SEV encryption:

- *`security.sev`* (page 437) : (bool) is SEV enabled for this VM
- *`security.sev.policy.es`* (page 437) : (bool) is SEV-ES enabled for this VM
- *`security.sev.session.dh`* (page 438) : (string) guest owner's `base64`-encoded Diffie-Hellman key
- *`security.sev.session.data`* (page 438) : (string) guest owner's `base64`-encoded session blob

`storage_pool_loop_resize`

This allows growing loop file backed storage pools by changing the `size` setting of the pool.

`migration_vm_live`

This adds support for performing VM QEMU to QEMU live migration for both shared storage (clustered Ceph) and non-shared storage pools.

This also adds the `CRIUType_VM_QEMU` value of `3` for the migration `CRIUType` `protobuf` field.

`ovn_nic_nesting`

This adds support for nesting an `ovn` NIC inside another `ovn` NIC on the same instance. This allows for an OVN logical switch port to be tunneled inside another OVN NIC using VLAN tagging.

This feature is configured by specifying the parent NIC name using the *nested* (page 465) property and the VLAN ID to use for tunneling with the *vlan* (page 466) property.

`oidc`

This adds support for OpenID Connect (OIDC) authentication.

This adds the following new configuration keys:

- *oidc.issuer* (page 406)
- *oidc.client.id* (page 406)
- *oidc.audience* (page 406)

`network_ovn_l3only`

This adds the ability to set an `ovn` network into "layer 3 only" mode. This mode can be enabled at IPv4 or IPv6 level using *ipv4.l3only* (page 590) and *ipv6.l3only* (page 591) configuration options respectively.

With this mode enabled the following changes are made to the network:

- The virtual router's internal port address will be configured with a single host netmask (e.g. /32 for IPv4 or /128 for IPv6).
- Static routes for active instance NIC addresses will be added to the virtual router.
- A discard route for the entire internal subnet will be added to the virtual router to prevent packets destined for inactive addresses from escaping to the uplink network.
- The DHCPv4 server will be configured to indicate that a netmask of 255.255.255.255 be used for instance configuration.

`ovn_nic_acceleration_vdpa`

This updates the `ovn_nic_acceleration` API extension. The *acceleration* (page 463) configuration key for OVN NICs can now takes the value `vdpa` to support Virtual Data Path Acceleration (VDPA).

`cluster_healing`

This adds cluster healing which automatically evacuates offline cluster members.

This adds the following new configuration key:

- *cluster.healing_threshold* (page 407)

The configuration key takes an integer, and can be disabled by setting it to 0 (default). If set, the value represents the threshold after which an offline cluster member is to be evacuated. In case the value is lower than *cluster.offline_threshold* (page 408), that value will be used instead.

When the offline cluster member is evacuated, only remote-backed instances will be migrated. Local instances will be ignored as there is no way of migrating them once the cluster member is offline.

`instances_state_total`

This extension adds a new `total` field to `InstanceStateDisk` and `InstanceStateMemory`, both part of the instance's state API.

`auth_user`

Add current user details to the main API endpoint.

This introduces:

- `auth_user_name`
- `auth_user_method`

`security_csm`

Introduce a new *`security.csm`* (page 433) configuration key to control the use of `CSM` (Compatibility Support Module) to allow legacy operating systems to be run in LXD VMs.

`instances_rebuild`

This extension adds the ability to rebuild an instance with the same origin image, alternate image or as empty. A new `POST /1.0/instances/<name>/rebuild?project=<project>` API endpoint has been added as well as a new CLI command *`lxc rebuild`* (page 870).

`numa_cpu_placement`

This adds the possibility to place a set of CPUs in a desired set of NUMA nodes.

This adds the following new configuration key:

- *`limits.cpu.nodes`* (page 421) : (string) comma-separated list of NUMA node IDs or NUMA node ID ranges to place the CPUs (chosen with a dynamic value of *`limits.cpu`* (page 421)) in.

`custom_volume_iso`

This adds the possibility to import ISO images as custom storage volumes.

This adds the `--type` flag to *`lxc storage volume import`* (page 905).

`network_allocations`

This adds the possibility to list a LXD deployment's network allocations.

Through the *`lxc network list-allocations`* (page 813) command and the `--project <PROJECT> | --all-projects` flags, you can list all the used IP addresses, hardware addresses (for instances), resource URIs and whether it uses NAT for each `instance`, `network`, `network forward` and `network load-balancer`.

storage_api_remote_volume_snapshot_copy

This allows copying storage volume snapshots to and from remotes.

zfs_delegate

This implements a new *zfs.delegate* (page 569) volume Boolean for volumes on a ZFS storage driver. When enabled and a suitable system is in use (requires ZFS 2.2 or higher), the ZFS dataset will be delegated to the container, allowing for its use through the `zfs` command line tool.

operations_get_query_all_projects

This introduces support for the `all-projects` query parameter for the GET API calls to both `/1.0/operations` and `/1.0/operations?recursion=1`. This parameter allows bypassing the project name filter.

metadata_configuration

Adds the `GET /1.0/metadata/configuration` API endpoint to retrieve the generated metadata configuration in a JSON format. The JSON structure adopts the structure `"configs" > `ENTITY` > `ENTITY_SECTION` > "keys" > [<CONFIG_OPTION_0>, <CONFIG_OPTION_1>, ...]`. Check the list of *configuration options* (page 401) to see which configuration options are included.

syslog_socket

This introduces a syslog socket that can receive syslog formatted log messages. These can be viewed in the events API and `lxc monitor`, and can be forwarded to Loki. To enable this feature, set *core.syslog_socket* (page 404) to `true`.

event_lifecycle_name_and_project

This adds the fields `Name` and `Project` to `lifecycle` events.

instances_nic_limits_priority

This introduces a new per-NIC `limits.priority` option that works with both cgroup1 and cgroup2 unlike the deprecated `limits.network.priority` instance setting, which only worked with cgroup1.

disk_initial_volume_configuration

This API extension provides the capability to set initial volume configurations for instance root devices. Initial volume configurations are prefixed with `initial.` and can be specified either through profiles or directly during instance initialization using the `--device` flag.

Note that these configuration are applied only at the time of instance creation and subsequent modifications have no effect on existing devices.

operation_wait

This API extension indicates that the `/1.0/operations/{id}/wait` endpoint exists on the server. This indicates to the client that the endpoint can be used to wait for an operation to complete rather than waiting for an operation event via the `/1.0/events` endpoint.

cluster_internal_custom_volume_copy

This extension adds support for copying and moving custom storage volumes within a cluster with a single API call. Calling `POST /1.0/storage-pools/<pool>/custom?target=<target>` will copy the custom volume specified in the `source` part of the request. Calling `POST /1.0/storage-pools/<pool>/custom/<volume>?target=<target>` will move the custom volume from the source, specified in the `source` part of the request, to the target.

disk_io_bus

This introduces a new *io.bus* (page 481) property to disk devices which can be used to override the bus the disk is attached to.

storage_cephfs_create_missing

This introduces the configuration keys *cephfs.create_missing* (page 527), *cephfs.osd_pg_num* (page 528), *cephfs.meta_pool* (page 528) and *cephfs.data_pool* (page 528) to be used when adding a `cephfs` storage pool to instruct LXD to create the necessary entities for the storage pool, if they do not exist.

instance_move_config

This API extension provides the ability to use flags `--profile`, `--no-profile`, `--device`, and `--config` when moving an instance between projects and/or storage pools.

ovn_ssl_config

This introduces new server configuration keys to provide the SSL CA and client key pair to access the OVN databases. The new configuration keys are *network.ovn.ca_cert* (page 412), *network.ovn.client_cert* (page 413) and *network.ovn.client_key* (page 413).

init_preseed_storage_volumes

This API extension provides the ability to configure storage volumes in preseed init.

metrics_instances_count

This extends the metrics to include the containers and virtual machines counts. Instances are counted irrespective of their state.

server_instance_type_info

This API extension enables querying a server's supported instance types. When querying the `/1.0` endpoint, a new field named `instance_types` is added to the retrieved data. This field indicates which instance types are supported by the server.

`resources_disk_mounted`

Adds a `mounted` field to disk resources that LXD discovers on the system, reporting whether that disk or partition is mounted.

`server_version_lts`

The API extension adds indication whether the LXD version is an LTS release. This is indicated when command `lxc version` is executed or when `/1.0` endpoint is queried.

`oidc_groups_claim`

This API extension enables setting an *`oidc.groups.claim`* (page 406) configuration key. If OIDC authentication is configured and this claim is set, LXD will request this claim in the scope of OIDC flow. The value of the claim will be extracted and might be used to make authorization decisions.

`loki_config_instance`

Adds a new *`loki.instance`* (page 410) server configuration key to customize the `instance` field in Loki events. This can be used to expose the name of the cluster rather than the individual system name sending the event as that's usually already covered by the `location` field.

`storage_volatile_uuid`

Adds a new `volatile.uuid` configuration key to all storage volumes, snapshots and buckets. This information can be used by storage drivers as a separate identifier besides the name when working with volumes.

`import_instance_devices`

This API extension provides the ability to use flags `--device` when importing an instance to override instance's devices.

`instances_uefi_vars`

This API extension indicates that the `/1.0/instances/{name}/uefi-vars` endpoint is supported on the server. This endpoint allows to get the full list of UEFI variables (HTTP method GET) or replace the entire set of UEFI variables (HTTP method PUT).

`instances_migration_stateful`

This API extension allows newly created VMs to have their *`migration.stateful`* (page 429) configuration key automatically set through the new server-level configuration key *`instances.migration.stateful`* (page 411). If `migration.stateful` is already set at the profile or instance level then `instances.migration.stateful` is not applied.

access_management

Adds new APIs under `/1.0/auth` for viewing and managing identities, groups, and permissions. Adds an embedded OpenFGA authorization driver for enforcing fine-grained permissions.

> **Important**
>
> Prior to the addition of this extension, all OIDC clients were given full access to LXD (equivalent to Unix socket access). This extension revokes access to all OIDC clients. To regain access, a user must:
>
> 1. Make a call to the OIDC enabled LXD remote (e.g. `lxc info`) to ensure that their OIDC identity is added to the LXD database.
> 2. Create a group: `lxc auth group create <group_name>`
> 3. Grant the group a suitable permission. As all OIDC clients prior to this extension have had full access to LXD, the corresponding permission is `admin` on `server`. To grant this permission to your group, run: `lxc auth group permission add <group_name> server admin`
> 4. Add themselves to the group. To do this, run: `lxc auth identity group add oidc/<email_address> <group_name>`
>
> Steps 2 to 4 above cannot be performed via OIDC authentication (access has been revoked). They must be performed by a sufficiently privileged user, either via Unix socket or unrestricted TLS client certificate.
>
> For more information on access control for OIDC clients, see *Fine-grained authorization* (page 364).

vm_disk_io_limits

Adds the ability to limit disk I/O for virtual machines.

storage_volumes_all

This API extension adds support for listing storage volumes from all storage pools via `/1.0/storage-volumes` or `/1.0/storage-volumes/{type}` to filter by volume type. Also adds a `pool` field to storage volumes.

instances_files_modify_permissions

Adds the ability for `POST /1.0/instances/{name}/files` to modify the permissions of files that already exist via the `X-LXD-modify-perm` header.

`X-LXD-modify-perm` should be a comma-separated list of 0 or more of `mode`, `uid`, and `gid`.

image_restriction_nesting

This extension adds a new image restriction, `requirements.nesting` which when `true` indicates that an image cannot be run without nesting.

container_syscall_intercept_finit_module

Adds the *linux.kernel_modules.load* (page 417) container configuration option. If the option is set to ondemand, the finit_modules() syscall is intercepted and a privileged user in the container's user namespace can load the Linux kernel modules specified in the allow list *linux.kernel_modules* (page 417).

device_usb_serial

This adds new configuration keys *serial* (page 491), *busnum* (page 490) and *devnum* (page 490) for *device type usb* (page 490). The feature has been added to make it possible to distinguish between devices with identical *vendorid* (page 491) and *productid* (page 490).

network_allocate_external_ips

Adds the ability to use an unspecified IPv4 (0.0.0.0) or IPv6 (::) address in the listen_address field of the request body for POST /1.0/networks/{networkName}/load-balancers and POST /1.0/networks/{networkName}/forwards. If an unspecified IP address is used, supported drivers will allocate an available listen address automatically. Allocation of external IP addresses is currently supported by the OVN network driver. The OVN driver will allocate IP addresses from the subnets specified in the uplink network's ipv4.routes and ipv6.routes configuration options.

explicit_trust_token

Adds the ability to explicitly specify a trust token when creating a certificate and joining an existing cluster.

shared_custom_block_volumes

This adds a configuration key security.shared to custom block volumes. If unset or false, the custom block volume cannot be attached to multiple instances. This feature was added to prevent data loss which can happen when custom block volumes are attached to multiple instances at once.

instance_import_conversion

Adds the ability to convert images from different formats (e.g. VMDK or QCow2) into RAW image format and import them as LXD instances.

instance_create_start

Adds a start field to the POST /1.0/instances API which when set to true will have the instance automatically start upon creation.

In this scenario, the creation and startup is part of a single background operation.

instance_protection_start

Enables setting the *security.protection.start* (page 437) field which prevents instances from being started if set to true.

devlxd_images_vm

Enables the *security.devlxd.images* (page 435) configuration option for virtual machines. This controls the availability of a `/1.0/images/FINGERPRINT/export` API over `devlxd`. This can be used by a virtual machine running LXD to access raw images from the host.

disk_io_bus_virtio_blk

Adds a new `virtio-blk` value for `io.bus` on `disk` devices which allows for the attached disk to be connected to the `virtio-blk` bus.

metrics_api_requests

Adds the following internal metrics:

- Total completed requests
- Number of ongoing requests

projects_limits_disk_pool

This introduces per-pool project disk limits, introducing a `limits.disk.pool.NAME` configuration option to the project limits. When `limits.disk.pool.POOLNAME: 0` for a project, the pool is excluded from `lxc storage list` in that project.

ubuntu_pro_guest_attach

Adds a new *ubuntu_pro.guest_attach* (page 417) configuration option for instances. When set to `on`, if the host has guest attachment enabled, the guest can request a guest token for Ubuntu Pro via `devlxd`.

metadata_configuration_entity_types

This adds entity type metadata to `GET /1.0/metadata/configuration`. The entity type metadata is a JSON object under the `entities` key.

access_management_tls

Expands APIs under `/1.0/auth` to include:

1. Creation of fine-grained TLS identities, whose permissions are managed via group membership. This is performed via `POST /1.0/auth/identities/tls`. If the request body contains `{"token": true}`, a token will be returned that may be used by a non-authenticated caller to gain trust with the LXD server (the caller must send their certificate during the TLS handshake). If the request body contains `{"certificate": "<base64 encoded x509 certificate>"}"`, the identity will be created directly. The request body may also specify an array of group names. The caller must have `can_create_identities` on `server`.
2. Deletion of OIDC and fine-grained TLS identities. This is performed via `DELETE /1.0/auth/identities/tls/{nameOrFingerprint}` or `DELETE /1.0/auth/identities/oidc/{nameOrEmailAddress}`. The caller must have `can_delete` on the identity. All identities may delete their own identity. For OIDC identities this revokes all access but does not

revoke trust (authentication is performed by the identity provider). For fine-grained TLS identities, this revokes all access and revokes trust.

3. Functionality to update the certificate of a fine-grained TLS identity. This is performed via `PUT /1.0/auth/identities/tls/{nameOrFingerprint}` or `PATCH /1.0/auth/identities/tls/{nameOrFingerprint}`. The caller must provide a base64 encoded x509 certificate in the `certificate` field of the request body. Fine-grained TLS identities may update their own certificate. To update the certificate of another identity, the caller must have `can_edit` on the identity.

`network_allocations_ovn_uplink`

Includes OVN virtual routers external IPs to `/1.0/network-allocations` responses with the type `uplink`. Introduces the `network` field on each allocation, indicating to which network each allocated address belongs. And lastly, adds a `project` field on leases, leases can be retrieved via `/1.0/networks/<network>/leases`.

`network_ovn_uplink_vlan`

Adds support for using a bridge network with a specified VLAN ID as an OVN uplink.

`state_logical_cpus`

Adds `logical_cpus` field to `GET /1.0/cluster/members/{name}/state` which contains the total available logical CPUs available when LXD started.

`vm_limits_cpu_pin_strategy`

Adds a new *`limits.cpu.pin_strategy`* (page 422) configuration option for virtual machines. This option controls the CPU pinning strategy. When set to `none`, CPU auto pinning is disabled. When set to `auto`, CPU auto pinning is enabled.

`gpu_cdi`

Adds support for using the Container Device Interface (CDI) specification to configure GPU passthrough in LXD containers. The `id` field of GPU devices now accepts CDI identifiers (for example, `{VENDOR_DOMAIN_NAME}/gpu=gpu{INDEX}`) for containers, in addition to DRM card IDs. This enables GPU passthrough for devices that don't use PCI addressing (like NVIDIA Tegra iGPUs) and provides a more flexible way to identify and configure GPU devices.

`images_all_projects`

This adds support for listing images across all projects using the `all-projects` parameter in `GET /1.0/images` requests.

`metadata_configuration_scope`

This adds scope metadata to `GET /1.0/metadata/configuration`. Options marked with a `global` scope are applied to all cluster members. Options with a `local` scope must be set on a per-member basis.

unix_device_hotplug_ownership_inherit

Adds a new *ownership.inherit* (page 504) configuration option for `unix-hotplug` devices. This option controls whether the device inherits ownership (GID and/or UID) from the host. When set to `true` and GID and/or UID are unset, host ownership is inherited. When set to `false`, host ownership is not inherited and ownership can be configured by setting *gid* (page 503) and *uid* (page 504).

unix_device_hotplug_subsystem_device_option

Adds a new *subsystem* (page 504) configuration option for `unix-hotplug` devices. This adds support for detecting `unix-hotplug` devices by subsystem, and can be used in conjunction with *productid* (page 504) and *vendorid* (page 504).

storage_ceph_osd_pool_size

This introduces the configuration keys *ceph.osd.pool_size* (page 537), and *cephfs.osd_pool_size* (page 528) to be used when adding or updating a `ceph` or `cephfs` storage pool to instruct LXD to create set the replication size for the underlying OSD pools.

network_get_target

Adds optional `target` parameter to `GET /1.0/network`. When target is set, forward the request to the specified cluster member and return the non-managed interfaces from that member.

network_zones_all_projects

This adds support for listing network zones across all projects using the `all-projects` parameter in `GET /1.0/network-zones` requests.

vm_root_volume_attachment

Adds support for virtual-machine root volumes and snapshots to be attached to other instances as disk devices. Introduces the `source.type` and `source.snapshot` keys for disk devices.

projects_limits_uplink_ips

Introduces per-project uplink IP limits for each available uplink network, adding `limits.networks.uplink_ips.ipv4.NETWORK_NAME` and `limits.networks.uplink_ips.ipv6.NETWORK_NAME` configuration keys for projects with `features.networks` enabled. These keys define the maximum value of IPs made available on a network named NETWORK_NAME to be assigned as uplink IPs for entities inside a certain project. These entities can be other networks, network forwards or load balancers.

entities_with_entitlements

Adds `fine_grained` field to `GET /1.0/auth/identities/current` to indicate if the current identity interacting with the LXD API is fine-grained (that is, associated permissions are managed via group membership). Allows LXD entities to be returned with an `access_entitlements` field if the current identity is fine-grained and the GET request to fetch the LXD entities has the with-access-entitlements=<comma_separated_list_of_candidate_entitlements> query parameter.

profiles_all_projects

This adds support for listing profiles across all projects using the all-projects parameter in GET /1.0/profiles requests.

storage_driver_powerflex

Adds a new powerflex storage driver which allows the consumption of storage volumes from a Dell PowerFlex storage array using NVMe/TCP and SDC. The following new pool level configuration keys have been added:

1. *powerflex.clone_copy* (page 543)
2. *powerflex.domain* (page 543)
3. *powerflex.gateway* (page 543)
4. *powerflex.gateway.verify* (page 544)
5. *powerflex.mode* (page 544)
6. *powerflex.pool* (page 544)
7. *powerflex.sdt* (page 544)
8. *powerflex.user.name* (page 544)
9. *powerflex.user.password* (page 545)

The following configuration keys have been added for volumes backed by PowerFlex:

1. *block.type* (page 546)

storage_driver_pure

Adds a new pure storage driver which allows the consumption of storage volumes from a Pure Storage storage array using either iSCSI or NVMe/TCP.

The following pool level configuration keys have been added:

1. *pure.gateway* (page 551)
2. *pure.gateway.verify* (page 551)
3. *pure.api.token* (page 551)
4. *pure.mode* (page 551)
5. *pure.target* (page 551)

cloud_init_ssh_keys

Adds support for injecting additional SSH public keys into instances through *cloud-init* (page 419) without conflicting with any configuration present on *cloud-init.vendor-data* (page 420) or *cloud-init.user-data* (page 420).

To achieve this, the `cloud-init.ssh-keys.KEYNAME` configuration key is added for both instances and profiles. This key is used to define a public key to be injected. `KEYNAME` can be any arbitrary name for the injected key.

The value for `cloud-init.ssh-keys.KEYNAME` should be `<user>:<key>`, where `<user>` is the name of the user for whom to inject the key. For `<key>`, provide either the public key or a `cloud-init` import ID for a key hosted elsewhere. Example valid values for `cloud-init.ssh-keys.KEYNAME` are `root:gh:githubUser` or `myUser:ssh-keyAlg base64PublicKey`.

`oidc_scopes`

This API extension enables setting an *`oidc.scopes`* (page 407) configuration key, which accepts a space-separated list of OIDC scopes to request from the identity provider. This configuration option can be used to request additional scopes that might be required for retrieving *identity provider groups* (page 367) from the identity provider. Additionally, the optional scopes `profile` and `offline_access` can be unset via this setting. Note that the `openid` and `email` scopes are always required.

`project_default_network_and_storage`

Adds flags –network and –storage. The –network flag adds a network device connected to the specified network to the default profile. The –storage flag adds a root disk device using the specified storage pool to the default profile.

`client_cert_presence`

Adds the field `client_certificate` to `GET /1.0` to indicate if the current request has a client certificate in it. This is for informational purposes only and does not affect the behavior of the API.

`clustering_groups_used_by`

This API extension adds a `used_by` field to the API response for a *cluster group* (page 372). Deletion of a cluster group is disallowed if the cluster group is referenced by project configuration (see *`restricted.cluster.groups`* (page 514)).

`container_bpf_delegation`

Adds new *`security.delegate_bpf`* (page 434).* group of options in order to support eBPF delegation using BPF Token mechanism. See *Privilege delegation using BPF Token* (page 381) for more information.

`override_snapshot_profiles_on_copy`

This adds a request option to set snapshot's target profile on instance copy to be inherited from target instance.

`resources_device_fs_uuid`

Adds the field `device_fs_uuid` including the respective UUID to each disk and partition indicating whether or not a filesystem is located on the device.

`backup_metadata_version`

Adds the field `version` when exporting instances and custom storage volumes to define the backup file format. In case the field is omitted, the server's default version is used. This maintains backwards compatibility with older clients.

When exporting an instance, the specific version can be provided using the `--export-version` flag:

```
lxc export v1 --export-version 2
```

The same applies when exporting a custom storage volume:

```
lxc storage volume export pool1 vol1 --export-version 2
```

`storage_buckets_all_projects`

This adds support for listing storage buckets across all projects using the `all-projects` parameter in `GET /1.0/storage-pools/POOL/buckets` requests.

`network_acls_all_projects`

This adds support for listing network ACLs across all projects using the `all-projects` parameter in `GET /1.0/network-acls` requests.

`networks_all_projects`

This adds support for listing networks across all projects using the `all-projects` parameter in `GET /1.0/networks` requests.

`clustering_restore_skip_mode`

Adds a `skip` mode to the restore request. This mode restores a cluster member's status to `ONLINE` without restarting any of its stopped local instances or migrating back instances that were evacuated to other cluster members.

`disk_io_threads_virtiofsd`

Adds the *`io.threads`* (page 481) option on `disk` devices which is used to control the `virtiofsd` thread pool size when sharing file systems into VMs. This can help improve I/O performance.

`oidc_client_secret`

This adds support for the *`oidc.client.secret`* (page 406) configuration key. If set, the LXD server will use this value in the OpenID Connect (OIDC) authorization code flow, which is used by LXD UI. This configuration value is not shared with other LXD clients (such as the LXD CLI).

Events

Introduction

Events are messages about actions that have occurred over LXD. Using the API endpoint `/1.0/events` directly or via *lxc monitor* (page 787) will connect to a WebSocket through which logs and life-cycle messages will be streamed.

Event types

LXD Currently supports three event types.

- `logging`: Shows all logging messages regardless of the server logging level.
- `operation`: Shows all ongoing operations from creation to completion (including updates to their state and progress metadata).
- `lifecycle`: Shows an audit trail for specific actions occurring over LXD.

Event structure

Example

```yaml
location: cluster_name
metadata:
  action: network-updated
  requestor:
    protocol: unix
    username: root
  source: /1.0/networks/lxdbr0
timestamp: "2021-03-14T00:00:00Z"
type: lifecycle
```

- `location`: The cluster member name (if clustered).
- `timestamp`: Time that the event occurred in RFC3339 format.
- `type`: The type of event this is (one of `logging`, `operation`, or `lifecycle`).
- `metadata`: Information about the specific event type.

Logging event structure

- `message`: The log message.
- `level`: The log-level of the log.
- `context`: Additional information included in the event.

Operation event structure

- `id`: The UUID of the operation.
- `class`: The type of operation (`task`, `token`, or `websocket`).
- `description`: A description of the operation.
- `created_at`: The operation's creation date.

- `updated_at`: The operation's date of last change.
- `status`: The current state of the operation.
- `status_code`: The operation status code.
- `resources`: Resources affected by this operation.
- `metadata`: Operation specific metadata.
- `may_cancel`: Whether the operation may be canceled.
- `err`: Error message of the operation.
- `location`: The cluster member name (if clustered).

Life-cycle event structure

- `action`: The life-cycle action that occurred.
- `requestor`: Information about who is making the request (if applicable).
- `source`: Path to what is being acted upon.
- `context`: Additional information included in the event.

Supported life-cycle events

Name	Description
`certificate-created`	A new certificate has been added to the server trust store.
`certificate-deleted`	The certificate has been deleted from the trust store.
`certificate-updated`	The certificate's configuration has been updated.
`cluster-certificate-updated`	The certificate for the whole cluster has changed.
`cluster-disabled`	Clustering has been disabled for this machine.
`cluster-enabled`	Clustering has been enabled for this machine.
`cluster-group-created`	A new cluster group has been created.
`cluster-group-deleted`	A cluster group has been deleted.
`cluster-group-renamed`	A cluster group has been renamed.
`cluster-group-updated`	A cluster group has been updated.
`cluster-member-added`	A new machine has joined the cluster.
`cluster-member-removed`	The cluster member has been removed from the cluster.
`cluster-member-renamed`	The cluster member has been renamed.
`cluster-member-updated`	The cluster member's configuration been edited.
`cluster-token-created`	A join token for adding a cluster member has been created.
`config-updated`	The server configuration has changed.
`image-alias-created`	An alias has been created for an existing image.
`image-alias-deleted`	An alias has been deleted for an existing image.
`image-alias-renamed`	The alias for an existing image has been renamed.
`image-alias-updated`	The configuration for an image alias has changed.
`image-created`	A new image has been added to the image store.
`image-deleted`	The image has been deleted from the image store.
`image-refreshed`	The local image copy has updated to the current source image
`image-retrieved`	The raw image file has been downloaded from the server.
`image-secret-created`	A one-time key to fetch this image has been created.

Table 2 – continued fr

Name	Description
image-updated	The image's configuration has changed.
instance-backup-created	A backup of the instance has been created.
instance-backup-deleted	The instance backup has been deleted.
instance-backup-renamed	The instance backup has been renamed.
instance-backup-retrieved	The raw instance backup file has been downloaded.
instance-console	Connected to the console of the instance.
instance-console-reset	The console buffer has been reset.
instance-console-retrieved	The console log has been downloaded.
instance-created	A new instance has been created.
instance-deleted	The instance has been deleted.
instance-exec	A command has been executed on the instance.
instance-file-deleted	A file on the instance has been deleted.
instance-file-pushed	The file has been pushed to the instance.
instance-file-retrieved	The file has been downloaded from the instance.
instance-log-deleted	The instance's specified log file has been deleted.
instance-log-retrieved	The instance's specified log file has been downloaded.
instance-metadata-retrieved	The instance's image metadata has been downloaded.
instance-metadata-template-created	A new image template file for the instance has been created.
instance-metadata-template-deleted	The image template file for the instance has been deleted.
instance-metadata-template-retrieved	The image template file for the instance has been downloaded
instance-metadata-updated	The instance's image metadata has changed.
instance-paused	The instance has been put in a paused state.
instance-ready	The instance is ready.
instance-renamed	The instance has been renamed.
instance-restarted	The instance has restarted.
instance-restored	The instance has been restored from a snapshot.
instance-resumed	The instance has resumed after being paused.
instance-shutdown	The instance has shut down.
instance-snapshot-created	A snapshot of the instance has been created.
instance-snapshot-deleted	The instance snapshot has been deleted.
instance-snapshot-renamed	The instance snapshot has been renamed.
instance-snapshot-updated	The instance snapshot's configuration has changed.
instance-started	The instance has started.
instance-stopped	The instance has stopped.
instance-updated	The instance's configuration has changed.
network-acl-created	A new network ACL has been created.
network-acl-deleted	The network ACL has been deleted.
network-acl-renamed	The network ACL has been renamed.
network-acl-updated	The network ACL configuration has changed.
network-created	A network device has been created.
network-deleted	The network device has been deleted.
network-forward-created	A new network forward has been created.
network-forward-deleted	The network forward has been deleted.
network-forward-updated	The network forward has been updated.
network-peer-created	A new network peer has been created.
network-peer-deleted	The network peer has been deleted.
network-peer-updated	The network peer has been updated.
network-renamed	The network device has been renamed.

Name	Description
network-updated	The network device's configuration has changed.
network-zone-created	A new network zone has been created.
network-zone-deleted	The network zone has been deleted.
network-zone-record-created	A new network zone record has been created.
network-zone-record-deleted	The network zone record has been deleted.
network-zone-record-updated	The network zone record has been updated.
network-zone-updated	The network zone has been updated.
operation-cancelled	The operation has been canceled.
profile-created	A new profile has been created.
profile-deleted	The profile has been deleted.
profile-renamed	The profile has been renamed .
profile-updated	The profile's configuration has changed.
project-created	A new project has been created.
project-deleted	The project has been deleted.
project-renamed	The project has been renamed.
project-updated	The project's configuration has changed.
storage-pool-created	A new storage pool has been created.
storage-pool-deleted	The storage pool has been deleted.
storage-pool-updated	The storage pool's configuration has changed.
storage-volume-backup-created	A new backup for the storage volume has been created.
storage-volume-backup-deleted	The storage volume's backup has been deleted.
storage-volume-backup-renamed	The storage volume's backup has been renamed.
storage-volume-backup-retrieved	The storage volume's backup has been downloaded.
storage-volume-created	A new storage volume has been created.
storage-volume-deleted	The storage volume has been deleted.
storage-volume-renamed	The storage volume has been renamed.
storage-volume-restored	The storage volume has been restored from a snapshot.
storage-volume-snapshot-created	A new storage volume snapshot has been created.
storage-volume-snapshot-deleted	The storage volume's snapshot has been deleted.
storage-volume-snapshot-renamed	The storage volume's snapshot has been renamed.
storage-volume-snapshot-updated	The configuration for the storage volume's snapshot has chang
storage-volume-updated	The storage volume's configuration has changed.
warning-acknowledged	The warning's status has been set to "acknowledged".
warning-deleted	The warning has been deleted.
warning-reset	The warning's status has been set to "new".

Communication between instance and host

Communication between the hosted workload (instance) and its host while not strictly needed is a pretty useful feature.

In LXD, this feature is implemented through a `/dev/lxd/sock` node which is created and set up for all LXD instances.

This file is a Unix socket which processes inside the instance can connect to. It's multi-threaded so multiple clients can be connected at the same time.

> **Note**
>
> *security.devlxd* (page 435) must be set to `true` (which is the default) for an instance to allow access to the socket.

Implementation details

LXD on the host binds `/var/lib/lxd/devlxd/sock` and starts listening for new connections on it.

This socket is then exposed into every single instance started by LXD at `/dev/lxd/sock`.

The single socket is required so we can exceed 4096 instances, otherwise, LXD would have to bind a different socket for every instance, quickly reaching the FD limit.

Authentication

Queries on `/dev/lxd/sock` will only return information related to the requesting instance. To figure out where a request comes from, LXD will extract the initial socket's user credentials and compare that to the list of instances it manages.

Protocol

The protocol on `/dev/lxd/sock` is plain-text HTTP with JSON messaging, so very similar to the local version of the LXD protocol.

Unlike the main LXD API, there is no background operation and no authentication support in the `/dev/lxd/sock` API.

REST-API

API structure

- /
  - `/1.0`
    - `/1.0/config`
      - `/1.0/config/{key}`
    - `/1.0/devices`
    - `/1.0/events`
    - `/1.0/images/{fingerprint}/export`
    - `/1.0/meta-data`

API details

/

GET

- Description: List of supported APIs

- Return: list of supported API endpoint URLs (by default `['/1.0']`)

Return value:

```
[
    "/1.0"
]
```

`/1.0`

GET

- Description: Information about the 1.0 API
- Return: JSON object

Return value:

```
{
    "api_version": "1.0",
    "location": "foo.example.com",
    "instance_type": "container",
    "state": "Started",
}
```

PATCH

- Description: Update instance state (valid states are `Ready` and `Started`)
- Return: none

Input:

```
{
   "state": "Ready"
}
```

`/1.0/config`

GET

- Description: List of configuration keys
- Return: list of configuration keys URL

Note that the configuration key names match those in the instance configuration, however not all configuration namespaces will be exported to `/dev/lxd/sock`. Currently only the `cloud-init.*` and `user.*` keys are accessible to the instance.

At this time, there also aren't any instance-writable namespace.

Return value:

```
[
    "/1.0/config/user.a"
]
```

/1.0/config/<KEY>

GET

- Description: Value of that key
- Return: Plain-text value

Return value:

```
blah
```

/1.0/devices

GET

- Description: Map of instance devices
- Return: JSON object

Return value:

```
{
    "eth0": {
        "name": "eth0",
        "network": "lxdbr0",
        "type": "nic"
    },
    "root": {
        "path": "/",
        "pool": "default",
        "type": "disk"
    }
}
```

/1.0/events

GET

- Description: WebSocket upgrade
- Return: none (never ending flow of events)

Supported arguments are:

- type: comma-separated list of notifications to subscribe to (defaults to all)

The notification types are:

- `config` (changes to any of the `user.*` configuration keys)
- `device` (any device addition, change or removal)

This never returns. Each notification is sent as a separate JSON object:

```
{
    "timestamp": "2017-12-21T18:28:26.846603815-05:00",
```

(continues on next page)

(continued from previous page)

```
    "type": "device",
    "metadata": {
        "name": "kvm",
        "action": "added",
        "config": {
            "type": "unix-char",
            "path": "/dev/kvm"
        }
    }
}
```

```
{
    "timestamp": "2017-12-21T18:28:26.846603815-05:00",
    "type": "config",
    "metadata": {
        "key": "user.foo",
        "old_value": "",
        "value": "bar"
    }
}
```

`/1.0/images/<FINGERPRINT>/export`

GET

- Description: Download a public/cached image from the host
- Return: raw image or error
- Access: Requires *security.devlxd.images* (page 435) set to `true`

Return value:

```
See /1.0/images/<FINGERPRINT>/export in the daemon API.
```

`/1.0/meta-data`

GET

- Description: Container meta-data compatible with cloud-init
- Return: cloud-init meta-data

Return value:

```
instance-id: af6a01c7-f847-4688-a2a4-37fddd744625
local-hostname: abc
```

`/1.0/ubuntu-pro`

GET

- Description: Get Ubuntu Pro guest attachment setting for the instance
- Return: JSON object

Return value


```
{
    "guest_attach": "on"
}
```


`/1.0/ubuntu-pro/token`

POST

- Description: Get an Ubuntu Pro guest attachment token
- Return: JSON object

Return value


```
{
    "expires": "2025-03-23T20:00:00-04:00",
    "token": "<RANDOM-STRING>",
    "id": "9f65c3d0-c326-491e-927f-9b062b6649a0"
}
```


Related topics

How-to guides:

- *LXD server and client* (page 44)

Explanation:

- *About `lxd` and `lxc`* (page 345)
- *The LXD Dqlite database* (page 356)

4.6. Man pages

`lxc` is the command line client for LXD. Its usage is documented in the help pages for the `lxc` commands and subcommands.

4.6.1. Man pages

`lxc`

Command line client for LXD

Synopsis

Description: Command line client for LXD

All of LXD's features can be driven through the various commands below. For help with any of those, simply call them with –help.

Options

```
      --all            Show less common commands
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc alias* (page 692) - Manage command aliases
- *lxc auth* (page 695) - Manage user authorization
- *lxc cluster* (page 715) - Manage cluster members
- *lxc completion* (page 733) - Generate the autocompletion script for the specified shell
- *lxc config* (page 738) - Manage instance and server configuration options
- *lxc console* (page 761) - Attach to instance consoles
- *lxc copy* (page 761) - Copy instances within or in between LXD servers
- *lxc delete* (page 762) - Delete instances and snapshots
- *lxc exec* (page 763) - Execute commands in instances
- *lxc export* (page 764) - Export instance backups
- *lxc file* (page 765) - Manage files in instances
- *lxc image* (page 770) - Manage images
- *lxc import* (page 781) - Import instance backups
- *lxc info* (page 782) - Show instance or server information
- *lxc init* (page 783) - Create instances from images
- *lxc launch* (page 784) - Create and start instances from images
- *lxc list* (page 785) - List instances
- *lxc manpage* (page 787) - Generate manpages for all commands
- *lxc monitor* (page 787) - Monitor a local or remote LXD server
- *lxc move* (page 788) - Move instances within or in between LXD servers
- *lxc network* (page 790) - Manage and attach instances to networks
- *lxc operation* (page 844) - List, show and delete background operations
- *lxc pause* (page 847) - Pause instances
- *lxc profile* (page 847) - Manage profiles
- *lxc project* (page 861) - Manage projects

- *lxc publish* (page 869) - Publish instances as images
- *lxc query* (page 869) - Send a raw query to LXD
- *lxc rebuild* (page 870) - Rebuild instances
- *lxc remote* (page 871) - Manage the list of remote servers
- *lxc rename* (page 876) - Rename instances and snapshots
- *lxc restart* (page 876) - Restart instances
- *lxc restore* (page 877) - Restore instances from snapshots
- *lxc snapshot* (page 878) - Create instance snapshots
- *lxc start* (page 879) - Start instances
- *lxc stop* (page 879) - Stop instances
- *lxc storage* (page 880) - Manage storage pools and volumes
- *lxc version* (page 913) - Show local and remote versions
- *lxc warning* (page 914) - Manage warnings

`lxc alias`

Manage command aliases

Synopsis

Description: Manage command aliases

```
lxc alias [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD
- *lxc alias add* (page 693) - Add new aliases
- *lxc alias list* (page 693) - List aliases
- *lxc alias remove* (page 694) - Remove aliases
- *lxc alias rename* (page 694) - Rename aliases

lxc alias add

Add new aliases

Synopsis

Description: Add new aliases

```
lxc alias add <alias> <target> [flags]
```

Examples

```
  lxc alias add list "list -c ns46S"
      Overwrite the "list" command to pass -c ns46S.
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc alias* (page 692) - Manage command aliases

lxc alias list

List aliases

Synopsis

Description: List aliases

```
lxc alias list [flags]
```

Options

```
  -f, --format   Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
```

(continues on next page)

(continued from previous page)

```
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc alias* (page 692) - Manage command aliases

`lxc alias remove`

Remove aliases

Synopsis

Description: Remove aliases

```
lxc alias remove <alias> [flags]
```

Examples

```
  lxc alias remove my-list
      Remove the "my-list" alias.
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc alias* (page 692) - Manage command aliases

`lxc alias rename`

Rename aliases

Synopsis

Description: Rename aliases

```
lxc alias rename <old alias> <new alias> [flags]
```

Examples

```
lxc alias rename list my-list
    Rename existing alias "list" to "my-list".
```

Options inherited from parent commands

```
      --debug            Show all debug messages
      --force-local      Force using the local unix socket
  -h, --help             Print help
      --project          Override the source project
  -q, --quiet            Don't show progress information
      --sub-commands     Use with help or --help to view sub-commands
  -v, --verbose          Show all information messages
      --version          Print version number
```

SEE ALSO

- *lxc alias* (page 692) - Manage command aliases

`lxc auth`

Manage user authorization

Synopsis

Description: Manage user authorization

```
lxc auth [flags]
```

Options inherited from parent commands

```
      --debug            Show all debug messages
      --force-local      Force using the local unix socket
  -h, --help             Print help
      --project          Override the source project
  -q, --quiet            Don't show progress information
      --sub-commands     Use with help or --help to view sub-commands
  -v, --verbose          Show all information messages
      --version          Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD
- *lxc auth group* (page 696) - Manage groups
- *lxc auth identity* (page 702) - Manage identities
- *lxc auth identity-provider-group* (page 708) - Manage groups
- *lxc auth permission* (page 714) - Inspect permissions

`lxc auth group`

Manage groups

Synopsis

Description: Manage groups

```
lxc auth group [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth* (page 695) - Manage user authorization
- *lxc auth group create* (page 696) - Create groups
- *lxc auth group delete* (page 697) - Delete groups
- *lxc auth group edit* (page 698) - Edit groups as YAML
- *lxc auth group list* (page 698) - List groups
- *lxc auth group permission* (page 699) - Manage permissions
- *lxc auth group rename* (page 701) - Rename groups
- *lxc auth group show* (page 701) - Show group configurations

`lxc auth group create`

Create groups

Synopsis

Description: Create groups

```
lxc auth group create [<remote>:]<group> [flags]
```

Options

```
  -d, --description string   Group description
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth group* (page 696) - Manage groups

`lxc auth group delete`

Delete groups

Synopsis

Description: Delete groups

```
lxc auth group delete [<remote>:]<group> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth group* (page 696) - Manage groups

`lxc auth group edit`

Edit groups as YAML

Synopsis

Description: Edit groups as YAML

```
lxc auth group edit [<remote>:]<group> [flags]
```

Examples

```
  lxc auth group edit <group> < group.yaml
     Update a group using the content of group.yaml
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth group* (page 696) - Manage groups

`lxc auth group list`

List groups

Synopsis

Description: List groups

```
lxc auth group list [<remote>:] [flags]
```

Options

```
  -f, --format   Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth group* (page 696) - Manage groups

`lxc auth group permission`

Manage permissions

Synopsis

Description: Manage permissions

```
lxc auth group permission [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth group* (page 696) - Manage groups
- *lxc auth group permission add* (page 699) - Add permissions to groups
- *lxc auth group permission remove* (page 700) - Remove permissions from groups

`lxc auth group permission add`

Add permissions to groups

Synopsis

Description: Add permissions to groups

```
lxc auth group permission add [<remote>:]<group> <entity_type> [<entity_name>]
<entitlement> [<key>=<value>...] [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth group permission* (page 699) - Manage permissions

`lxc auth group permission remove`

Remove permissions from groups

Synopsis

Description: Remove permissions from groups

```
lxc auth group permission remove [<remote>:]<group> <entity_type> [<entity_name>]
<entitlement> [<key>=<value>...] [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth group permission* (page 699) - Manage permissions

lxc auth group rename

Rename groups

Synopsis

Description: Rename groups

```
lxc auth group rename [<remote>:]<group> <new_name> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth group* (page 696) - Manage groups

lxc auth group show

Show group configurations

Synopsis

Description: Show group configurations

```
lxc auth group show [<remote>:]<group> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth group* (page 696) - Manage groups

`lxc auth identity`

Manage identities

Synopsis

Description: Manage identities

```
lxc auth identity [flags]
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc auth* (page 695) - Manage user authorization
- *lxc auth identity create* (page 702) - Create an identity
- *lxc auth identity delete* (page 703) - Delete an identity
- *lxc auth identity edit* (page 704) - Edit an identity as YAML
- *lxc auth identity group* (page 705) - Manage groups for the identity
- *lxc auth identity info* (page 706) - View the current identity
- *lxc auth identity list* (page 707) - List identities
- *lxc auth identity show* (page 707) - View an identity

`lxc auth identity create`

Create an identity

Synopsis

Description: Create a TLS identity

```
lxc auth identity create [<remote>:]<authentication_method>/<name> [<path to PEM
encoded certificate>] [[--group <group_name>]] [flags]
```

Options

```
  -g, --group strings   Groups to add to the identity
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc auth identity* (page 702) - Manage identities

`lxc auth identity delete`

Delete an identity

Synopsis

Description: Delete an identity

```
lxc auth identity delete [<remote>:]<authentication_method>/<name_or_identifier>
[flags]
```

Examples

```
  lxc auth identity delete oidc/jane.doe@example.com
          Delete the OIDC identity with email address "jane.doe@example.com" in
the default remote.

  lxc auth identity delete oidc/'Jane Doe'
          Delete the OIDC identity with name "Jane Doe" in the default remote
(there must be only one OIDC identity on the server with this name).

  lxc auth identity delete my-remote:tls/
12beaccbf9e7b7445185581b70099a5962c927e85006d5883856d909fe79f976
          Delete the TLS identity with certificate fingerprint
"12beaccbf9e7b7445185581b70099a5962c927e85006d5883856d909fe79f976" in remote "my-
remote".

  lxc auth identity delete my-remote:tls/jane-doe
          Delete the TLS identity with name "jane-doe" in remote "my-remote"
(there must be only one TLS identity on "my-remote" with this name).
```

Options inherited from parent commands

```
      --debug            Show all debug messages
      --force-local      Force using the local unix socket
  -h, --help             Print help
      --project          Override the source project
  -q, --quiet            Don't show progress information
      --sub-commands     Use with help or --help to view sub-commands
  -v, --verbose          Show all information messages
      --version          Print version number
```

SEE ALSO

- *lxc auth identity* (page 702) - Manage identities

`lxc auth identity edit`

Edit an identity as YAML

Synopsis

Description: Edit an identity as YAML

```
lxc auth identity edit [<remote>:]<group> [flags]
```

Examples

```
  lxc auth identity edit <authentication_method>/<name_or_identifier> < identity.
yaml
     Update an identity using the content of identity.yaml
```

Options inherited from parent commands

```
      --debug            Show all debug messages
      --force-local      Force using the local unix socket
  -h, --help             Print help
      --project          Override the source project
  -q, --quiet            Don't show progress information
      --sub-commands     Use with help or --help to view sub-commands
  -v, --verbose          Show all information messages
      --version          Print version number
```

SEE ALSO

- *lxc auth identity* (page 702) - Manage identities

`lxc auth identity group`

Manage groups for the identity

Synopsis

Description: Manage groups for the identity

```
lxc auth identity group [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth identity* (page 702) - Manage identities
- *lxc auth identity group add* (page 705) - Add a group to an identity
- *lxc auth identity group remove* (page 706) - Remove a group from an identity

`lxc auth identity group add`

Add a group to an identity

Synopsis

Description: Add a group to an identity

```
lxc auth identity group add [<remote>:]<authentication_method>/<name_or_
identifier> <group> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth identity group* (page 705) - Manage groups for the identity

lxc auth identity group remove

Remove a group from an identity

Synopsis

Description: Remove a group from an identity

```
lxc auth identity group remove [<remote>:]<authentication_method>/<name_or_
identifier> <group> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth identity group* (page 705) - Manage groups for the identity

lxc auth identity info

View the current identity

Synopsis

Description: Show the current identity

This command will display permissions for the current user. This includes contextual information, such as effective groups and permissions that are granted via identity provider group mappings.

```
lxc auth identity info [<remote>:] [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
```

(continues on next page)

(continued from previous page)

```
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth identity* (page 702) - Manage identities

`lxc auth identity list`

List identities

Synopsis

Description: List identities

```
lxc auth identity list [<remote>:] [flags]
```

Options

```
  -f, --format   Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth identity* (page 702) - Manage identities

`lxc auth identity show`

View an identity

Synopsis

Description: Show identity configurations

The argument must be a concatenation of the authentication method and either the name or identifier of the identity, delimited by a forward slash. This command will fail if an identity name is used that is not unique within the authentication method. Use the identifier instead if this occurs.

```
lxc auth identity show [<remote>:]<authentication_method>/<name_or_identifier>
[flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth identity* (page 702) - Manage identities

lxc auth identity-provider-group

Manage groups

Synopsis

Description: Manage groups

```
lxc auth identity-provider-group [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth* (page 695) - Manage user authorization
- *lxc auth identity-provider-group create* (page 709) - Create identity provider groups
- *lxc auth identity-provider-group delete* (page 709) - Delete identity provider groups
- *lxc auth identity-provider-group edit* (page 710) - Edit identity provider groups as YAML

- *lxc auth identity-provider-group group* (page 710) - Manage identity provider group mappings
- *lxc auth identity-provider-group list* (page 712) - List identity provider groups
- *lxc auth identity-provider-group rename* (page 713) - Rename identity provider groups
- *lxc auth identity-provider-group show* (page 713) - Show an identity provider group

`lxc auth identity-provider-group create`

Create identity provider groups

Synopsis

Description: Create identity provider groups

```
lxc auth identity-provider-group create [<remote>:]<group> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth identity-provider-group* (page 708) - Manage groups

`lxc auth identity-provider-group delete`

Delete identity provider groups

Synopsis

Description: Delete identity provider groups

```
lxc auth identity-provider-group delete [<remote>:]<identity_provider_group>
[flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
```

(continues on next page)

```
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth identity-provider-group* (page 708) - Manage groups

`lxc auth identity-provider-group edit`

Edit identity provider groups as YAML

Synopsis

Description: Edit identity provider groups as YAML

```
lxc auth identity-provider-group edit [<remote>:]<identity_provider_group> [flags]
```

Examples

```
  lxc auth identity-provider-group edit <identity_provider_group> < identity-
provider-group.yaml
     Update an identity provider group using the content of identity-provider-
group.yaml
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth identity-provider-group* (page 708) - Manage groups

`lxc auth identity-provider-group group`

Manage identity provider group mappings

Synopsis

Description: Manage identity provider group mappings

```
lxc auth identity-provider-group group [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth identity-provider-group* (page 708) - Manage groups
- *lxc auth identity-provider-group group add* (page 711) - Add a group to an identity provider group
- *lxc auth identity-provider-group group remove* (page 712) - Remove identities from groups

`lxc auth identity-provider-group group add`

Add a group to an identity provider group

Synopsis

Description: Add a group to an identity provider group

```
lxc auth identity-provider-group group add [<remote>:]<identity_provider_group>
<group> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth identity-provider-group group* (page 710) - Manage identity provider group mappings

lxc auth identity-provider-group group remove

Remove identities from groups

Synopsis

Description: Remove identities from groups

```
lxc auth identity-provider-group group remove [<remote>:]<authentication_method>/
<name_or_identifier> <group> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth identity-provider-group group* (page 710) - Manage identity provider group mappings

lxc auth identity-provider-group list

List identity provider groups

Synopsis

Description: List identity provider groups

```
lxc auth identity-provider-group list [<remote>:] [flags]
```

Options

```
  -f, --format   Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth identity-provider-group* (page 708) - Manage groups

`lxc auth identity-provider-group rename`

Rename identity provider groups

Synopsis

Description: Rename identity provider groups

```
lxc auth identity-provider-group rename [<remote>:]<identity_provider_group> <new_
name> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth identity-provider-group* (page 708) - Manage groups

`lxc auth identity-provider-group show`

Show an identity provider group

Synopsis

Description: Show an identity provider group

```
lxc auth identity-provider-group show [<remote>:]<identity_provider_group> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth identity-provider-group* (page 708) - Manage groups

`lxc auth permission`

Inspect permissions

Synopsis

Description: Inspect permissions

```
lxc auth permission [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc auth* (page 695) - Manage user authorization
- *lxc auth permission list* (page 714) - List permissions

`lxc auth permission list`

List permissions

Synopsis

Description: List permissions

```
lxc auth permission list [<remote>:] [project=<project_name>] [entity_type=
<entity_type>] [flags]
```

Options

```
  -f, --format string          Display format (json, yaml, table, compact, csv)
(default "table")
      --max-entitlements int   Maximum number of unassigned entitlements to
display before overflowing (set to zero to display all) (default 3)
```

Options inherited from parent commands

```
      --debug            Show all debug messages
      --force-local      Force using the local unix socket
  -h, --help             Print help
      --project          Override the source project
  -q, --quiet            Don't show progress information
      --sub-commands     Use with help or --help to view sub-commands
  -v, --verbose          Show all information messages
      --version          Print version number
```

SEE ALSO

- *lxc auth permission* (page 714) - Inspect permissions

`lxc cluster`

Manage cluster members

Synopsis

Description: Manage cluster members

```
lxc cluster [flags]
```

Options inherited from parent commands

```
      --debug            Show all debug messages
      --force-local      Force using the local unix socket
  -h, --help             Print help
      --project          Override the source project
  -q, --quiet            Don't show progress information
      --sub-commands     Use with help or --help to view sub-commands
  -v, --verbose          Show all information messages
      --version          Print version number
```

SEE ALSO

`lxc cluster add`

Request a join token for adding a cluster member

Synopsis

Description: Request a join token for adding a cluster member

```
lxc cluster add [[<remote>:]<member>] [flags]
```

Options

```
      --name   Cluster member name (alternative to passing it as an argument)
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc cluster* (page 715) - Manage cluster members

lxc cluster edit

Edit cluster member configurations as YAML

Synopsis

Description: Edit cluster member configurations as YAML

```
lxc cluster edit [<remote>:]<member> [flags]
```

Examples

```
  lxc cluster edit <cluster member> < member.yaml
      Update a cluster member using the content of member.yaml
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc cluster* (page 715) - Manage cluster members

lxc cluster enable

Enable clustering on a single non-clustered LXD server

Synopsis

Description: Enable clustering on a single non-clustered LXD server

```
This command turns a non-clustered LXD server into the first member of a new
LXD cluster, which will have the given name.

It's required that LXD is already available on the network. You can check
this by running 'lxc config get core.https_address'. If either an IP address
and port is displayed, or both, LXD is already available on the network. If
no value is set, use 'lxc config set core.https_address' to set it.
```

```
lxc cluster enable [<remote>:] <name> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc cluster* (page 715) - Manage cluster members

`lxc cluster evacuate`

Evacuate cluster member

Synopsis

Description: Evacuate cluster member

Evacuation actions:

- stop: stop all instances on the member
- migrate: migrate all instances on the member to other members
- live-migrate: live migrate all instances on the member to other members

```
lxc cluster evacuate [<remote>:]<member> [flags]
```

Options

```
      --action   Force a particular instance evacuation action. One of stop,
migrate or live-migrate
      --force    Force evacuation without user confirmation
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc cluster* (page 715) - Manage cluster members

`lxc cluster get`

Get values for cluster member configuration keys

Synopsis

Description: Get values for cluster member configuration keys

```
lxc cluster get [<remote>:]<member> <key> [flags]
```

Options

```
  -p, --property   Get the key as a cluster property
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc cluster* (page 715) - Manage cluster members

`lxc cluster group`

Manage cluster groups

Synopsis

Description: Manage cluster groups

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc cluster* (page 715) - Manage cluster members
- *lxc cluster group add* (page 720) - Add member to group
- *lxc cluster group assign* (page 721) - Assign sets of groups to cluster members
- *lxc cluster group create* (page 721) - Create a cluster group
- *lxc cluster group delete* (page 722) - Delete a cluster group
- *lxc cluster group edit* (page 723) - Edit a cluster group
- *lxc cluster group list* (page 723) - List all the cluster groups
- *lxc cluster group remove* (page 724) - Remove member from group
- *lxc cluster group rename* (page 724) - Rename a cluster group
- *lxc cluster group show* (page 725) - Show cluster group configurations

`lxc cluster group add`

Add member to group

Synopsis

Description: Add a cluster member to a cluster group

```
lxc cluster group add [<remote>:]<member> <group> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
```

(continues on next page)

(continued from previous page)

```
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc cluster group* (page 719) - Manage cluster groups

`lxc cluster group assign`

Assign sets of groups to cluster members

Synopsis

Description: Assign sets of groups to cluster members

```
lxc cluster group assign [<remote>:]<member> <group> [flags]
```

Examples

```
  lxc cluster group assign foo default,bar
      Set the groups for "foo" to "default" and "bar".

  lxc cluster group assign foo default
      Reset "foo" to only using the "default" cluster group.
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc cluster group* (page 719) - Manage cluster groups

`lxc cluster group create`

Create a cluster group

Synopsis

Description: Create a cluster group

```
lxc cluster group create [<remote>:]<group> [flags]
```

Examples

```
lxc cluster group create g1

lxc cluster group create g1 < config.yaml
        Create a cluster group with configuration from config.yaml
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc cluster group* (page 719) - Manage cluster groups

`lxc cluster group delete`

Delete a cluster group

Synopsis

Description: Delete a cluster group

```
lxc cluster group delete [<remote>:]<group> [flags]
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc cluster group* (page 719) - Manage cluster groups

lxc cluster group edit

Edit a cluster group

Synopsis

Description: Edit a cluster group

```
lxc cluster group edit [<remote>:]<group> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc cluster group* (page 719) - Manage cluster groups

lxc cluster group list

List all the cluster groups

Synopsis

Description: List all the cluster groups

```
lxc cluster group list [<remote>:] [flags]
```

Options

```
  -f, --format   Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
```

(continues on next page)

```
  -q, --quiet         Don't show progress information
      --sub-commands  Use with help or --help to view sub-commands
  -v, --verbose       Show all information messages
      --version       Print version number
```

SEE ALSO

- *lxc cluster group* (page 719) - Manage cluster groups

`lxc cluster group remove`

Remove member from group

Synopsis

Description: Remove a cluster member from a cluster group

```
lxc cluster group remove [<remote>:]<member> <group> [flags]
```

Options inherited from parent commands

```
      --debug         Show all debug messages
      --force-local   Force using the local unix socket
  -h, --help          Print help
      --project       Override the source project
  -q, --quiet         Don't show progress information
      --sub-commands  Use with help or --help to view sub-commands
  -v, --verbose       Show all information messages
      --version       Print version number
```

SEE ALSO

- *lxc cluster group* (page 719) - Manage cluster groups

`lxc cluster group rename`

Rename a cluster group

Synopsis

Description: Rename a cluster group

```
lxc cluster group rename [<remote>:]<group> <new-name> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc cluster group* (page 719) - Manage cluster groups

`lxc cluster group show`

Show cluster group configurations

Synopsis

Description: Show cluster group configurations

```
lxc cluster group show [<remote>:]<group> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc cluster group* (page 719) - Manage cluster groups

`lxc cluster info`

Show useful information about a cluster member

Synopsis

Description: Show useful information about a cluster member

```
lxc cluster info [<remote>:]<member> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc cluster* (page 715) - Manage cluster members

`lxc cluster list`

List all the cluster members

Synopsis

Description: List all the cluster members

```
lxc cluster list [<remote>:] [flags]
```

Options

```
  -f, --format   Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc cluster* (page 715) - Manage cluster members

`lxc cluster list-tokens`

List all active cluster member join tokens

Synopsis

Description: List all active cluster member join tokens

```
lxc cluster list-tokens [<remote>:] [flags]
```

Options

```
  -f, --format   Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc cluster* (page 715) - Manage cluster members

`lxc cluster remove`

Remove a member from the cluster

Synopsis

Description: Remove a member from the cluster

```
lxc cluster remove [<remote>:]<member> [flags]
```

Options

```
  -f, --force   Force removing a member, even if degraded
      --yes     Don't require user confirmation for using --force
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
```

(continues on next page)

(continued from previous page)

```
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc cluster* (page 715) - Manage cluster members

`lxc cluster rename`

Rename a cluster member

Synopsis

Description: Rename a cluster member

```
lxc cluster rename [<remote>:]<member> <new-name> [flags]
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc cluster* (page 715) - Manage cluster members

`lxc cluster restore`

Restore cluster member

Synopsis

Description: Restore cluster member

```
lxc cluster restore [<remote>:]<member> [flags]
```

Options

```
      --action   Force a particular instance restore action. Use "skip" to restore
only the cluster member status without starting local instances or migrating back
evacuated instances
      --force    Force restoration without user confirmation
```

Options inherited from parent commands

```
      --debug            Show all debug messages
      --force-local      Force using the local unix socket
  -h, --help             Print help
      --project          Override the source project
  -q, --quiet            Don't show progress information
      --sub-commands     Use with help or --help to view sub-commands
  -v, --verbose          Show all information messages
      --version          Print version number
```

SEE ALSO

- *lxc cluster* (page 715) - Manage cluster members

`lxc cluster revoke-token`

Revoke cluster member join token

Synopsis

Description: Revoke cluster member join token

```
lxc cluster revoke-token [<remote>:]<member> [flags]
```

Options inherited from parent commands

```
      --debug            Show all debug messages
      --force-local      Force using the local unix socket
  -h, --help             Print help
      --project          Override the source project
  -q, --quiet            Don't show progress information
      --sub-commands     Use with help or --help to view sub-commands
  -v, --verbose          Show all information messages
      --version          Print version number
```

SEE ALSO

- *lxc cluster* (page 715) - Manage cluster members

`lxc cluster role`

Manage cluster roles

Synopsis

Description: Manage cluster roles

```
lxc cluster role [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc cluster* (page 715) - Manage cluster members
- *lxc cluster role add* (page 730) - Add roles to a cluster member
- *lxc cluster role remove* (page 730) - Remove roles from a cluster member

`lxc cluster role add`

Add roles to a cluster member

Synopsis

Description: Add roles to a cluster member

```
lxc cluster role add [<remote>:]<member> <role[,role...]> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc cluster role* (page 729) - Manage cluster roles

`lxc cluster role remove`

Remove roles from a cluster member

Synopsis

Description: Remove roles from a cluster member

```
lxc cluster role remove [<remote>:]<member> <role[,role...]> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc cluster role* (page 729) - Manage cluster roles

lxc cluster set

Set a cluster member's configuration keys

Synopsis

Description: Set a cluster member's configuration keys

```
lxc cluster set [<remote>:]<member> <key>=<value>... [flags]
```

Options

```
  -p, --property   Set the key as a cluster property
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc cluster* (page 715) - Manage cluster members

`lxc cluster show`

Show details of a cluster member

Synopsis

Description: Show details of a cluster member

```
lxc cluster show [<remote>:]<member> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc cluster* (page 715) - Manage cluster members

`lxc cluster unset`

Unset a cluster member's configuration keys

Synopsis

Description: Unset a cluster member's configuration keys

```
lxc cluster unset [<remote>:]<member> <key> [flags]
```

Options

```
  -p, --property   Unset the key as a cluster property
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
```

(continues on next page)

```
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc cluster* (page 715) - Manage cluster members

lxc cluster update-certificate

Update cluster certificate

Synopsis

Description: Update cluster certificate with PEM certificate and key read from input files.

```
lxc cluster update-certificate [<remote>:] <cert.crt> <cert.key> [flags]
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc cluster* (page 715) - Manage cluster members

lxc completion

Generate the autocompletion script for the specified shell

Synopsis

Generate the autocompletion script for lxc for the specified shell. See each sub-command's help for details on how to use the generated script.

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
```

(continues on next page)

```
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD
- *lxc completion bash* (page 734) - Generate the autocompletion script for bash
- *lxc completion fish* (page 735) - Generate the autocompletion script for fish
- *lxc completion powershell* (page 736) - Generate the autocompletion script for powershell
- *lxc completion zsh* (page 737) - Generate the autocompletion script for zsh

`lxc completion bash`

Generate the autocompletion script for bash

Synopsis

Generate the autocompletion script for the bash shell.

This script depends on the 'bash-completion' package. If it is not installed already, you can install it via your OS's package manager.

To load completions in your current shell session:

```
source <(lxc completion bash)
```

To load completions for every new session, execute once:

Linux:

```
lxc completion bash > /etc/bash_completion.d/lxc
```

macOS:

```
lxc completion bash > $(brew --prefix)/etc/bash_completion.d/lxc
```

You will need to start a new shell for this setup to take effect.

```
lxc completion bash
```

Options

```
      --no-descriptions   disable completion descriptions
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc completion* (page 733) - Generate the autocompletion script for the specified shell

`lxc completion fish`

Generate the autocompletion script for fish

Synopsis

Generate the autocompletion script for the fish shell.

To load completions in your current shell session:

```
lxc completion fish | source
```

To load completions for every new session, execute once:

```
lxc completion fish > ~/.config/fish/completions/lxc.fish
```

You will need to start a new shell for this setup to take effect.

```
lxc completion fish [flags]
```

Options

```
      --no-descriptions   disable completion descriptions
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
```

(continues on next page)

```
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc completion* (page 733) - Generate the autocompletion script for the specified shell

`lxc completion powershell`

Generate the autocompletion script for powershell

Synopsis

Generate the autocompletion script for powershell.

To load completions in your current shell session:

```
lxc completion powershell | Out-String | Invoke-Expression
```

To load completions for every new session, add the output of the above command to your powershell profile.

```
lxc completion powershell [flags]
```

Options

```
    --no-descriptions   disable completion descriptions
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc completion* (page 733) - Generate the autocompletion script for the specified shell

lxc completion zsh

Generate the autocompletion script for zsh

Synopsis

Generate the autocompletion script for the zsh shell.

If shell completion is not already enabled in your environment you will need to enable it. You can execute the following once:

```
echo "autoload -U compinit; compinit" >> ~/.zshrc
```

To load completions in your current shell session:

```
source <(lxc completion zsh)
```

To load completions for every new session, execute once:

Linux:

```
lxc completion zsh > "${fpath[1]}/_lxc"
```

macOS:

```
lxc completion zsh > $(brew --prefix)/share/zsh/site-functions/_lxc
```

You will need to start a new shell for this setup to take effect.

```
lxc completion zsh [flags]
```

Options

```
      --no-descriptions   disable completion descriptions
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc completion* (page 733) - Generate the autocompletion script for the specified shell

`lxc config`

Manage instance and server configuration options

Synopsis

Description: Manage instance and server configuration options

```
lxc config [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD
- *lxc config device* (page 738) - Manage devices
- *lxc config edit* (page 744) - Edit instance or server configurations as YAML
- *lxc config get* (page 745) - Get values for instance or server configuration keys
- *lxc config metadata* (page 745) - Manage instance metadata files
- *lxc config set* (page 747) - Set instance or server configuration keys
- *lxc config show* (page 748) - Show instance or server configurations
- *lxc config template* (page 748) - Manage instance file templates
- *lxc config trust* (page 752) - Manage trusted clients
- *lxc config uefi* (page 756) - Manage instance UEFI variables
- *lxc config unset* (page 760) - Unset instance or server configuration keys

`lxc config device`

Manage devices

Synopsis

Description: Manage devices

```
lxc config device [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config* (page 738) - Manage instance and server configuration options
- *lxc config device add* (page 739) - Add instance devices
- *lxc config device get* (page 740) - Get values for device configuration keys
- *lxc config device list* (page 741) - List instance devices
- *lxc config device override* (page 741) - Copy profile inherited devices and override configuration keys
- *lxc config device remove* (page 742) - Remove instance devices
- *lxc config device set* (page 742) - Set device configuration keys
- *lxc config device show* (page 743) - Show full device configuration
- *lxc config device unset* (page 743) - Unset device configuration keys

`lxc config device add`

Add instance devices

Synopsis

Description: Add instance devices

```
lxc config device add [<remote>:]<instance> <device> <type> [key=value...] [flags]
```

Examples

```
  lxc config device add [<remote>:]instance1 <device-name> disk source=/share/c1
path=/opt
      Will mount the host's /share/c1 onto /opt in the instance.
```

(continues on next page)

(continued from previous page)

```
  lxc config device add [<remote>:]instance1 <device-name> disk pool=some-pool
source=some-volume path=/opt
      Will mount the some-volume volume on some-pool onto /opt in the instance.
```

Options inherited from parent commands

```
      --debug            Show all debug messages
      --force-local      Force using the local unix socket
  -h, --help             Print help
      --project          Override the source project
  -q, --quiet            Don't show progress information
      --sub-commands     Use with help or --help to view sub-commands
  -v, --verbose          Show all information messages
      --version          Print version number
```

SEE ALSO

- *lxc config device* (page 738) - Manage devices

`lxc config device get`

Get values for device configuration keys

Synopsis

Description: Get values for device configuration keys

```
lxc config device get [<remote>:]<instance> <device> <key> [flags]
```

Options inherited from parent commands

```
      --debug            Show all debug messages
      --force-local      Force using the local unix socket
  -h, --help             Print help
      --project          Override the source project
  -q, --quiet            Don't show progress information
      --sub-commands     Use with help or --help to view sub-commands
  -v, --verbose          Show all information messages
      --version          Print version number
```

SEE ALSO

- *lxc config device* (page 738) - Manage devices

lxc config device list

List instance devices

Synopsis

Description: List instance devices

```
lxc config device list [<remote>:]<instance> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config device* (page 738) - Manage devices

lxc config device override

Copy profile inherited devices and override configuration keys

Synopsis

Description: Copy profile inherited devices and override configuration keys

```
lxc config device override [<remote>:]<instance> <device> [key=value...] [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config device* (page 738) - Manage devices

`lxc config device remove`

Remove instance devices

Synopsis

Description: Remove instance devices

```
lxc config device remove [<remote>:]<instance> <name>... [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config device* (page 738) - Manage devices

`lxc config device set`

Set device configuration keys

Synopsis

Description: Set device configuration keys

For backward compatibility, a single configuration key may still be set with: lxc config device set [:]

```
lxc config device set [<remote>:]<instance> <device> <key>=<value>... [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
```

(continues on next page)

(continued from previous page)

```
-v, --verbose         Show all information messages
    --version         Print version number
```

SEE ALSO

- *lxc config device* (page 738) - Manage devices

`lxc config device show`

Show full device configuration

Synopsis

Description: Show full device configuration

```
lxc config device show [<remote>:]<instance> [flags]
```

Options inherited from parent commands

```
    --debug           Show all debug messages
    --force-local     Force using the local unix socket
-h, --help            Print help
    --project         Override the source project
-q, --quiet           Don't show progress information
    --sub-commands    Use with help or --help to view sub-commands
-v, --verbose         Show all information messages
    --version         Print version number
```

SEE ALSO

- *lxc config device* (page 738) - Manage devices

`lxc config device unset`

Unset device configuration keys

Synopsis

Description: Unset device configuration keys

```
lxc config device unset [<remote>:]<instance> <device> <key> [flags]
```

Options inherited from parent commands

```
    --debug           Show all debug messages
    --force-local     Force using the local unix socket
-h, --help            Print help
    --project         Override the source project
```

(continues on next page)

(continued from previous page)

```
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config device* (page 738) - Manage devices

`lxc config edit`

Edit instance or server configurations as YAML

Synopsis

Description: Edit instance or server configurations as YAML

```
lxc config edit [<remote>:][<instance>[/<snapshot>]] [flags]
```

Examples

```
  lxc config edit <instance> < instance.yaml
      Update the instance configuration from config.yaml.
```

Options

```
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config* (page 738) - Manage instance and server configuration options

lxc config get

Get values for instance or server configuration keys

Synopsis

Description: Get values for instance or server configuration keys

```
lxc config get [<remote>:][<instance>] <key> [flags]
```

Options

```
  -e, --expanded   Access the expanded configuration
  -p, --property   Get the key as an instance property
      --target     Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config* (page 738) - Manage instance and server configuration options

lxc config metadata

Manage instance metadata files

Synopsis

Description: Manage instance metadata files

```
lxc config metadata [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
```

(continues on next page)

(continued from previous page)

```
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc config* (page 738) - Manage instance and server configuration options
- *lxc config metadata edit* (page 746) - Edit instance metadata files
- *lxc config metadata show* (page 746) - Show instance metadata files

`lxc config metadata edit`

Edit instance metadata files

Synopsis

Description: Edit instance metadata files

```
lxc config metadata edit [<remote>:]<instance> [flags]
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc config metadata* (page 745) - Manage instance metadata files

`lxc config metadata show`

Show instance metadata files

Synopsis

Description: Show instance metadata files

```
lxc config metadata show [<remote>:]<instance> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config metadata* (page 745) - Manage instance metadata files

lxc config set

Set instance or server configuration keys

Synopsis

Description: Set instance or server configuration keys

For backward compatibility, a single configuration key may still be set with: lxc config set [:][]

```
lxc config set [<remote>:][<instance>] <key>=<value>... [flags]
```

Examples

```
  lxc config set [<remote>:]<instance> limits.cpu=2
      Will set a CPU limit of "2" for the instance.

  lxc config set core.https_address=[::]:8443
      Will have LXD listen on IPv4 and IPv6 port 8443.
```

Options

```
  -p, --property   Set the key as an instance property
      --target     Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config* (page 738) - Manage instance and server configuration options

`lxc config show`

Show instance or server configurations

Synopsis

Description: Show instance or server configurations

```
lxc config show [<remote>:][<instance>[/<snapshot>]] [flags]
```

Options

```
  -e, --expanded   Show the expanded configuration
      --target     Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config* (page 738) - Manage instance and server configuration options

`lxc config template`

Manage instance file templates

Synopsis

Description: Manage instance file templates

```
lxc config template [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
```

(continues on next page)

(continued from previous page)

```
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config* (page 738) - Manage instance and server configuration options
- *lxc config template create* (page 749) - Create new instance file templates
- *lxc config template delete* (page 749) - Delete instance file templates
- *lxc config template edit* (page 750) - Edit instance file templates
- *lxc config template list* (page 751) - List instance file templates
- *lxc config template show* (page 751) - Show content of instance file templates

`lxc config template create`

Create new instance file templates

Synopsis

Description: Create new instance file templates

```
lxc config template create [<remote>:]<instance> <template> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config template* (page 748) - Manage instance file templates

`lxc config template delete`

Delete instance file templates

Synopsis

Description: Delete instance file templates

```
lxc config template delete [<remote>:]<instance> <template> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config template* (page 748) - Manage instance file templates

`lxc config template edit`

Edit instance file templates

Synopsis

Description: Edit instance file templates

```
lxc config template edit [<remote>:]<instance> <template> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config template* (page 748) - Manage instance file templates

`lxc config template list`

List instance file templates

Synopsis

Description: List instance file templates

```
lxc config template list [<remote>:]<instance> [flags]
```

Options

```
  -f, --format   Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config template* (page 748) - Manage instance file templates

`lxc config template show`

Show content of instance file templates

Synopsis

Description: Show content of instance file templates

```
lxc config template show [<remote>:]<instance> <template> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config template* (page 748) - Manage instance file templates

`lxc config trust`

Manage trusted clients

Synopsis

Description: Manage trusted clients

```
lxc config trust [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config* (page 738) - Manage instance and server configuration options
- *lxc config trust add* (page 752) - Add new trusted client
- *lxc config trust edit* (page 753) - Edit trust configurations as YAML
- *lxc config trust list* (page 754) - List trusted clients
- *lxc config trust list-tokens* (page 754) - List all active certificate add tokens
- *lxc config trust remove* (page 755) - Remove trusted client
- *lxc config trust revoke-token* (page 755) - Revoke certificate add token
- *lxc config trust show* (page 756) - Show trust configurations

`lxc config trust add`

Add new trusted client

Synopsis

Description: Add new trusted client

The following certificate types are supported:

- client (default)
- metrics

If the certificate is omitted, a token will be generated and returned. A client providing a valid token will have its client certificate added to the trusted list and the consumed token will be invalidated. Similar to certificates, tokens can be restricted to one or more projects.

```
lxc config trust add [<remote>:] [<cert>] [flags]
```

Options

```
      --name         Alternative certificate name
      --projects     List of projects to restrict the certificate to
      --restricted   Restrict the certificate to one or more projects
      --type         Type of certificate (default "client")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config trust* (page 752) - Manage trusted clients

`lxc config trust edit`

Edit trust configurations as YAML

Synopsis

Description: Edit trust configurations as YAML

```
lxc config trust edit [<remote>:]<fingerprint> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config trust* (page 752) - Manage trusted clients

`lxc config trust list`

List trusted clients

Synopsis

Description: List trusted clients

```
lxc config trust list [<remote>:] [flags]
```

Options

```
  -f, --format   Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config trust* (page 752) - Manage trusted clients

`lxc config trust list-tokens`

List all active certificate add tokens

Synopsis

Description: List all active certificate add tokens

```
lxc config trust list-tokens [<remote>:] [flags]
```

Options

```
  -f, --format   Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config trust* (page 752) - Manage trusted clients

`lxc config trust remove`

Remove trusted client

Synopsis

Description: Remove trusted client

```
lxc config trust remove [<remote>:]<fingerprint> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config trust* (page 752) - Manage trusted clients

`lxc config trust revoke-token`

Revoke certificate add token

Synopsis

Description: Revoke certificate add token

```
lxc config trust revoke-token [<remote>:] <name> [flags]
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc config trust* (page 752) - Manage trusted clients

`lxc config trust show`

Show trust configurations

Synopsis

Description: Show trust configurations

```
lxc config trust show [<remote>:]<fingerprint> [flags]
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc config trust* (page 752) - Manage trusted clients

`lxc config uefi`

Manage instance UEFI variables

Synopsis

Description: Manage instance UEFI variables

```
lxc config uefi [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config* (page 738) - Manage instance and server configuration options
- *lxc config uefi edit* (page 757) - Edit instance UEFI variables
- *lxc config uefi get* (page 758) - Get UEFI variables for instance
- *lxc config uefi set* (page 758) - Set UEFI variables for instance
- *lxc config uefi show* (page 759) - Show instance UEFI variables
- *lxc config uefi unset* (page 759) - Unset UEFI variables for instance

`lxc config uefi edit`

Edit instance UEFI variables

Synopsis

Description: Edit instance UEFI variables

```
lxc config uefi edit [<remote>:]<instance> [flags]
```

Examples

```
  lxc config uefi edit <instance> < instance_uefi_vars.yaml
      Set the instance UEFI variables from instance_uefi_vars.yaml.
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config uefi* (page 756) - Manage instance UEFI variables

`lxc config uefi get`

Get UEFI variables for instance

Synopsis

Description: Get UEFI variables for instance

```
lxc config uefi get [<remote>:]<instance> <key> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config uefi* (page 756) - Manage instance UEFI variables

`lxc config uefi set`

Set UEFI variables for instance

Synopsis

Description: Set UEFI variables for instance

```
lxc config uefi set [<remote>:]<instance> <key>=<value>... [flags]
```

Examples

```
  lxc config uefi set [<remote>:]<instance> testvar-9073e4e0-60ec-4b6e-9903-
4c223c260f3c=aabb
      Set a UEFI variable with name "testvar", GUID 9073e4e0-60ec-4b6e-9903-
4c223c260f3c and value "aabb" (HEX-encoded) for the instance.
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config uefi* (page 756) - Manage instance UEFI variables

`lxc config uefi show`

Show instance UEFI variables

Synopsis

Description: Show instance UEFI variables

```
lxc config uefi show [<remote>:]<instance> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc config uefi* (page 756) - Manage instance UEFI variables

`lxc config uefi unset`

Unset UEFI variables for instance

Synopsis

Description: Unset UEFI variables for instance

```
lxc config uefi unset [<remote>:]<instance> <key> [flags]
```

Options inherited from parent commands

```
      --debug            Show all debug messages
      --force-local      Force using the local unix socket
  -h, --help             Print help
      --project          Override the source project
  -q, --quiet            Don't show progress information
      --sub-commands     Use with help or --help to view sub-commands
  -v, --verbose          Show all information messages
      --version          Print version number
```

SEE ALSO

- *lxc config uefi* (page 756) - Manage instance UEFI variables

`lxc config unset`

Unset instance or server configuration keys

Synopsis

Description: Unset instance or server configuration keys

```
lxc config unset [<remote>:][<instance>] <key> [flags]
```

Options

```
  -p, --property   Unset the key as an instance property
      --target     Cluster member name
```

Options inherited from parent commands

```
      --debug            Show all debug messages
      --force-local      Force using the local unix socket
  -h, --help             Print help
      --project          Override the source project
  -q, --quiet            Don't show progress information
      --sub-commands     Use with help or --help to view sub-commands
  -v, --verbose          Show all information messages
      --version          Print version number
```

SEE ALSO

- *lxc config* (page 738) - Manage instance and server configuration options

lxc console

Attach to instance consoles

Synopsis

Description: Attach to instance consoles

This command allows you to interact with the boot console of an instance as well as retrieve past log entries from it.

```
lxc console [<remote>:]<instance> [flags]
```

Options

```
      --show-log   Retrieve the container's console log
  -t, --type       Type of connection to establish: 'console' for serial console,
'vga' for SPICE graphical output (default "console")
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

lxc copy

Copy instances within or in between LXD servers

Synopsis

Description: Copy instances within or in between LXD servers

Transfer modes (–mode):

- pull: Target server pulls the data from the source server (source must listen on network)
- push: Source server pushes the data to the target server (target must listen on network)
- relay: The CLI connects to both source and server and proxies the data (both source and target must listen on network)

The pull transfer mode is the default as it is compatible with all LXD versions.

```
lxc copy [<remote>:]<source>[/<snapshot>] [[<remote>:]<destination>] [flags]
```

Options

```
      --allow-inconsistent   Ignore copy errors for volatile files
  -c, --config               Config key/value to apply to the new instance
  -d, --device               New key/value to apply to a specific device
  -e, --ephemeral            Ephemeral instance
      --instance-only        Copy the instance without its snapshots
      --mode                 Transfer mode. One of pull, push or relay (default
"pull")
      --no-profiles          Create the instance with no profiles applied
  -p, --profile              Profile to apply to the new instance
      --refresh              Perform an incremental copy
      --stateless            Copy a stateful instance stateless
  -s, --storage              Storage pool name
      --target               Cluster member name
      --target-project       Copy to a project different from the source
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

`lxc delete`

Delete instances and snapshots

Synopsis

Description: Delete instances and snapshots

```
lxc delete [<remote>:]<instance>[/<snapshot>] [[<remote>:]<instance>[/<snapshot>].
..] [flags]
```

Options

```
-f, --force         Force the removal of running instances
-i, --interactive   Require user confirmation
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

lxc exec

Execute commands in instances

Synopsis

Description: Execute commands in instances

The command is executed directly using exec, so there is no shell and shell patterns (variables, file redirects, …) won't be understood. If you need a shell environment you need to execute the shell executable, passing the shell commands as arguments, for example:

```
lxc exec <instance> -- sh -c "cd /tmp && pwd"
```

Mode defaults to non-interactive, interactive mode is selected if both stdin AND stdout are terminals (stderr is ignored).

```
lxc exec [<remote>:]<instance> [flags] [--] <command line>
```

Options

```
    --cwd                    Directory to run the command in (default /root)
-n, --disable-stdin          Disable stdin (reads from /dev/null)
    --env                    Environment variable to set (e.g. HOME=/home/foo)
-t, --force-interactive      Force pseudo-terminal allocation
-T, --force-noninteractive   Disable pseudo-terminal allocation
    --group                  Group ID to run the command as (default 0)
    --mode                   Override the terminal mode (auto, interactive or
non-interactive) (default "auto")
    --user                   User ID to run the command as (default 0)
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

lxc export

Export instance backups

Synopsis

Description: Export instances as backup tarballs.

```
lxc export [<remote>:]<instance> [target] [--instance-only] [--optimized-storage]
[flags]
```

Examples

```
  lxc export u1 backup0.tar.gz
      Download a backup tarball of the u1 instance.
```

Options

```
      --compression         Compression algorithm to use (none for uncompressed)
      --export-version      Use a different metadata format version than the
latest one supported by the server (to support imports on older LXD versions)
      --instance-only       Whether or not to only backup the instance (without
snapshots)
      --optimized-storage   Use storage driver optimized format (can only be
restored on a similar pool)
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
```

(continues on next page)

(continued from previous page)

```
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

lxc file

Manage files in instances

Synopsis

Description: Manage files in instances

```
lxc file [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD
- *lxc file create* (page 765) - Create files and directories in instances
- *lxc file delete* (page 766) - Delete files in instances
- *lxc file edit* (page 767) - Edit files in instances
- *lxc file mount* (page 767) - Mount files from instances
- *lxc file pull* (page 768) - Pull files from instances
- *lxc file push* (page 769) - Push files into instances

lxc file create

Create files and directories in instances

Synopsis

Description: Create files and directories in instances

```
lxc file create [<remote>:]<instance>/<path> [<symlink target path>] [flags]
```

Examples

```
  lxc file create foo/bar
             To create a file /bar in the foo instance.
  lxc file create --type=symlink foo/bar baz
             To create a symlink /bar in instance foo whose target is baz.
```

Options

```
  -p, --create-dirs   Create any directories necessary
  -f, --force         Force creating files or directories
      --gid           Set the file's gid on create (default -1)
      --mode          Set the file's perms on create
      --type          The type to create (file, symlink, or directory) (default
"file")
      --uid           Set the file's uid on create (default -1)
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc file* (page 765) - Manage files in instances

`lxc file delete`

Delete files in instances

Synopsis

Description: Delete files in instances

```
lxc file delete [<remote>:]<instance>/<path> [[<remote>:]<instance>/<path>...]
[flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc file* (page 765) - Manage files in instances

`lxc file edit`

Edit files in instances

Synopsis

Description: Edit files in instances

```
lxc file edit [<remote>:]<instance>/<path> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc file* (page 765) - Manage files in instances

`lxc file mount`

Mount files from instances

Synopsis

Description: Mount files from instances

```
lxc file mount [<remote>:]<instance>[/<path>] [<target path>] [flags]
```

Examples

```
  lxc file mount foo/root fooroot
     To mount /root from the instance foo onto the local fooroot directory.
```

Options

```
      --auth-user string   Set authentication user when using SSH SFTP listener
      --listen string      Setup SSH SFTP listener on address:port instead of
mounting
      --no-auth            Disable authentication when using SSH SFTP listener
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc file* (page 765) - Manage files in instances

`lxc file pull`

Pull files from instances

Synopsis

Description: Pull files from instances

```
lxc file pull [<remote>:]<instance>/<path> [[<remote>:]<instance>/<path>...]
<target path> [flags]
```

Examples

```
  lxc file pull foo/etc/hosts .
     To pull /etc/hosts from the instance and write it to the current directory.
```

Options

```
  -p, --create-dirs   Create any directories necessary
  -r, --recursive     Recursively transfer files
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc file* (page 765) - Manage files in instances

`lxc file push`

Push files into instances

Synopsis

Description: Push files into instances

```
lxc file push <source path>... [<remote>:]<instance>/<path> [flags]
```

Examples

```
  lxc file push /etc/hosts foo/etc/hosts
     To push /etc/hosts into the instance "foo".
```

Options

```
  -p, --create-dirs   Create any directories necessary
      --gid           Set the file's gid on push (default -1)
      --mode          Set the file's perms on push
  -r, --recursive     Recursively transfer files
      --uid           Set the file's uid on push (default -1)
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc file* (page 765) - Manage files in instances

`lxc image`

Manage images

Synopsis

Description: Manage images

In LXD instances are created from images. Those images were themselves either generated from an existing instance or downloaded from an image server.

When using remote images, LXD will automatically cache images for you and remove them upon expiration.

The image unique identifier is the hash (sha-256) of its representation as a compressed tarball (or for split images, the concatenation of the metadata and rootfs tarballs).

Images can be referenced by their full hash, shortest unique partial hash or alias name (if one is set).

```
lxc image [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD
- *lxc image alias* (page 771) - Manage image aliases
- *lxc image copy* (page 774) - Copy images between servers
- *lxc image delete* (page 774) - Delete images
- *lxc image edit* (page 775) - Edit image properties
- *lxc image export* (page 776) - Export and download images
- *lxc image get-property* (page 776) - Get image properties
- *lxc image import* (page 777) - Import images into the image store
- *lxc image info* (page 777) - Show useful information about images

- *lxc image list* (page 778) - List images
- *lxc image refresh* (page 779) - Refresh images
- *lxc image set-property* (page 780) - Set image properties
- *lxc image show* (page 780) - Show image properties
- *lxc image unset-property* (page 781) - Unset image properties

`lxc image alias`

Manage image aliases

Synopsis

Description: Manage image aliases

```
lxc image alias [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc image* (page 770) - Manage images
- *lxc image alias create* (page 771) - Create aliases for existing images
- *lxc image alias delete* (page 772) - Delete image aliases
- *lxc image alias list* (page 772) - List image aliases
- *lxc image alias rename* (page 773) - Rename aliases

`lxc image alias create`

Create aliases for existing images

Synopsis

Description: Create aliases for existing images

```
lxc image alias create [<remote>:]<alias> <fingerprint> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc image alias* (page 771) - Manage image aliases

`lxc image alias delete`

Delete image aliases

Synopsis

Description: Delete image aliases

```
lxc image alias delete [<remote>:]<alias> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc image alias* (page 771) - Manage image aliases

`lxc image alias list`

List image aliases

Synopsis

Description: List image aliases

Filters may be part of the image hash or part of the image alias name.

```
lxc image alias list [<remote>:] [<filters>...] [flags]
```

Options

```
  -f, --format   Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc image alias* (page 771) - Manage image aliases

`lxc image alias rename`

Rename aliases

Synopsis

Description: Rename aliases

```
lxc image alias rename [<remote>:]<alias> <new-name> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc image alias* (page 771) - Manage image aliases

`lxc image copy`

Copy images between servers

Synopsis

Description: Copy images between servers

The auto-update flag instructs the server to keep this image up to date. It requires the source to be an alias and for it to be public.

```
lxc image copy [<remote>:]<image> <remote>: [flags]
```

Options

```
      --alias            New aliases to add to the image
      --auto-update      Keep the image up to date after initial copy
      --copy-aliases     Copy aliases from source
      --mode             Transfer mode. One of pull (default), push or relay
(default "pull")
  -p, --profile          Profile to apply to the new image
      --public           Make image public
      --target-project   Copy to a project different from the source
      --vm               Copy virtual machine images
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc image* (page 770) - Manage images

`lxc image delete`

Delete images

Synopsis

Description: Delete images

```
lxc image delete [<remote>:]<image> [[<remote>:]<image>...] [flags]
```

Options inherited from parent commands

```
      --debug            Show all debug messages
      --force-local      Force using the local unix socket
  -h, --help             Print help
      --project          Override the source project
  -q, --quiet            Don't show progress information
      --sub-commands     Use with help or --help to view sub-commands
  -v, --verbose          Show all information messages
      --version          Print version number
```

SEE ALSO

- *lxc image* (page 770) - Manage images

`lxc image edit`

Edit image properties

Synopsis

Description: Edit image properties

```
lxc image edit [<remote>:]<image> [flags]
```

Examples

```
  lxc image edit <image>
      Launch a text editor to edit the properties

  lxc image edit <image> < image.yaml
      Load the image properties from a YAML file
```

Options inherited from parent commands

```
      --debug            Show all debug messages
      --force-local      Force using the local unix socket
  -h, --help             Print help
      --project          Override the source project
  -q, --quiet            Don't show progress information
      --sub-commands     Use with help or --help to view sub-commands
  -v, --verbose          Show all information messages
      --version          Print version number
```

SEE ALSO

- *lxc image* (page 770) - Manage images

`lxc image export`

Export and download images

Synopsis

Description: Export and download images

The output target is optional and defaults to the working directory.

```
lxc image export [<remote>:]<image> [<target>] [flags]
```

Options

```
      --vm   Query virtual machine images
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc image* (page 770) - Manage images

`lxc image get-property`

Get image properties

Synopsis

Description: Get image properties

```
lxc image get-property [<remote>:]<image> <key> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
```

(continues on next page)

```
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc image* (page 770) - Manage images

`lxc image import`

Import images into the image store

Synopsis

Description: Import image into the image store

Directory import is only available on Linux and must be performed as root.

Descriptive properties can be set by providing key=value pairs. Example: os=Ubuntu release=noble variant=cloud.

```
lxc image import <tarball>|<directory>|<URL> [<rootfs tarball>] [<remote>:]
[key=value...] [flags]
```

Options

```
    --alias    New aliases to add to the image
    --public   Make image public
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc image* (page 770) - Manage images

`lxc image info`

Show useful information about images

Synopsis

Description: Show useful information about images

```
lxc image info [<remote>:]<image> [flags]
```

Options

```
      --vm   Query virtual machine images
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc image* (page 770) - Manage images

`lxc image list`

List images

Synopsis

Description: List images

Filters may be of the = form for property based filtering, or part of the image hash or part of the image alias name.

The -c option takes a (optionally comma-separated) list of arguments that control which image attributes to output when displaying in table or csv format.

Default column layout is: lfpdasu

Column shorthand chars:

```
  l - Shortest image alias (and optionally number of other aliases)
  L - Newline-separated list of all image aliases
  f - Fingerprint (short)
  F - Fingerprint (long)
  p - Whether image is public
  d - Description
  e - Project
  a - Architecture
```

(continues on next page)

```
  s - Size
  u - Upload date
  t - Type
```

```
lxc image list [<remote>:] [<filter>...] [flags]
```

Options

```
      --all-projects   Display images from all projects
  -c, --columns        Columns (default "lfpdatsu")
  -f, --format         Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc image* (page 770) - Manage images

`lxc image refresh`

Refresh images

Synopsis

Description: Refresh images

```
lxc image refresh [<remote>:]<image> [[<remote>:]<image>...] [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc image* (page 770) - Manage images

lxc image set-property

Set image properties

Synopsis

Description: Set image properties

```
lxc image set-property [<remote>:]<image> <key> <value> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc image* (page 770) - Manage images

lxc image show

Show image properties

Synopsis

Description: Show image properties

```
lxc image show [<remote>:]<image> [flags]
```

Options

```
      --vm   Query virtual machine images
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
```

(continues on next page)

```
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc image* (page 770) - Manage images

`lxc image unset-property`

Unset image properties

Synopsis

Description: Unset image properties

```
lxc image unset-property [<remote>:]<image> <key> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc image* (page 770) - Manage images

`lxc import`

Import instance backups

Synopsis

Description: Import backups of instances including their snapshots.

```
lxc import [<remote>:] <backup file> [<instance name>] [flags]
```

Examples

```
  lxc import backup0.tar.gz
      Create a new instance using backup0.tar.gz as the source.
```

Options

```
-d, --device    New key/value to apply to a specific device
-s, --storage   Storage pool name
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

`lxc info`

Show instance or server information

Synopsis

Description: Show instance or server information

```
lxc info [<remote>:][<instance>] [flags]
```

Examples

```
lxc info [<remote>:]<instance> [--show-log]
    For instance information.

lxc info [<remote>:] [--resources]
    For LXD server information.
```

Options

```
    --resources   Show the resources available to the server
    --show-log    Show the instance's last 100 log lines
    --target      Cluster member name
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
```

(continues on next page)

(continued from previous page)

```
  -h, --help          Print help
      --project       Override the source project
  -q, --quiet         Don't show progress information
      --sub-commands  Use with help or --help to view sub-commands
  -v, --verbose       Show all information messages
      --version       Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

`lxc init`

Create instances from images

Synopsis

Description: Create instances from images

```
lxc init [<remote>:]<image> [<remote>:][<name>] [flags]
```

Examples

```
  lxc init ubuntu:24.04 u1
      Create a container (but do not start it)

  lxc init ubuntu:24.04 u1 < config.yaml
      Create a container with configuration from config.yaml

  lxc init ubuntu:24.04 v1 --vm -c limits.cpu=4 -c limits.memory=4GiB
      Create a virtual machine with 4 vCPUs and 4GiB of RAM

  lxc init ubuntu:24.04 v1 --vm -c limits.cpu=2 -c limits.memory=8GiB -d root,
size=32GiB
      Create a virtual machine with 2 vCPUs, 8GiB of RAM and a root disk of 32GiB
```

Options

```
  -c, --config       Config key/value to apply to the new instance
  -d, --device       New key/value to apply to a specific device
      --empty        Create an empty instance
  -e, --ephemeral    Ephemeral instance
  -n, --network      Network name
      --no-profiles  Create the instance with no profiles applied
  -p, --profile      Profile to apply to the new instance
  -s, --storage      Storage pool name
      --target       Cluster member name
```

(continues on next page)

```
  -t, --type          Instance type
      --vm            Create a virtual machine
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

`lxc launch`

Create and start instances from images

Synopsis

Description: Create and start instances from images

```
lxc launch [<remote>:]<image> [<remote>:][<name>] [flags]
```

Examples

```
  lxc launch ubuntu:24.04 u1
      Create and start a container

  lxc launch ubuntu:24.04 u1 < config.yaml
      Create and start a container with configuration from config.yaml

  lxc launch ubuntu:24.04 u2 -t aws:t2.micro
      Create and start a container using the same size as an AWS t2.micro (1 vCPU,
1GiB of RAM)

  lxc launch ubuntu:24.04 v1 --vm -c limits.cpu=4 -c limits.memory=4GiB
      Create and start a virtual machine with 4 vCPUs and 4GiB of RAM

  lxc launch ubuntu:24.04 v1 --vm -c limits.cpu=2 -c limits.memory=8GiB -d root,
size=32GiB
      Create and start a virtual machine with 2 vCPUs, 8GiB of RAM and a root disk
of 32GiB
```

Options

```
-c, --config                Config key/value to apply to the new instance
    --console[="console"]   Immediately attach to the console
-d, --device                New key/value to apply to a specific device
    --empty                 Create an empty instance
-e, --ephemeral             Ephemeral instance
-n, --network               Network name
    --no-profiles           Create the instance with no profiles applied
-p, --profile               Profile to apply to the new instance
-s, --storage               Storage pool name
    --target                Cluster member name
-t, --type                  Instance type
    --vm                    Create a virtual machine
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

`lxc list`

List instances

Synopsis

Description: List instances

Default column layout: ns46tS Fast column layout: nsacPt

A single keyword like "web" which will list any instance with a name starting with "web". A regular expression on the instance name. (e.g. .*web.*01$). A key/value pair referring to a configuration item. For those, the namespace can be abbreviated to the smallest unambiguous identifier. A key/value pair where the key is a shorthand. Multiple values must be delimited by ','. Available shorthands: - type={instance type} - status={instance current lifecycle status} - architecture={instance architecture} - location={location name} - ipv4={ip or CIDR} - ipv6={ip or CIDR}

Examples: - "user.blah=abc" will list all instances with the "blah" user property set to "abc". - "u.blah=abc" will do the same - "security.privileged=true" will list all privileged instances - "s.privileged=true" will do the same - "type=container" will list all container instances - "type=container status=running" will list all running container instances

A regular expression matching a configuration item or its value. (e.g. volatile.eth0.hwaddr=00:16:3e:.*).

When multiple filters are passed, they are added one on top of the other, selecting instances which satisfy them all.

== Columns == The -c option takes a comma separated list of arguments that control which instance attributes to output when displaying in table or csv format.

Column arguments are either pre-defined shorthand chars (see below), or (extended) config keys.

Commas between consecutive shorthand chars are optional.

Pre-defined column shorthand chars: 4 - IPv4 address 6 - IPv6 address a - Architecture b - Storage pool c - Creation date d - Description D - disk usage e - Project name l - Last used date m - Memory usage M - Memory usage (%) n - Name N - Number of Processes p - PID of the instance's init process P - Profiles s - State S - Number of snapshots t - Type (container or virtual-machine, ephemeral indicated if applicable) u - CPU usage (in seconds) L - Location of the instance (e.g. its cluster member) f - Base Image Fingerprint (short) F - Base Image Fingerprint (long)

Custom columns are defined with "[config:|devices:]key[:name][:maxWidth]": KEY: The (extended) config or devices key to display. If [config:|devices:] is omitted then it defaults to config key. NAME: Name to display in the column header. Defaults to the key if not specified or empty.

```
MAXWIDTH: Max width of the column (longer results are truncated).
Defaults to -1 (unlimited). Use 0 to limit to the column header size.
```

```
lxc list [<remote>:] [<filter>...] [flags]
```

Examples

```
  lxc list -c nFs46,volatile.eth0.hwaddr:MAC,config:image.os,devices:eth0.
parent:ETHP
    Show instances using the "NAME", "BASE IMAGE", "STATE", "IPV4", "IPV6" and
"MAC" columns.
    "BASE IMAGE", "MAC" and "IMAGE OS" are custom columns generated from instance
configuration keys.
    "ETHP" is a custom column generated from a device key.

  lxc list -c ns,user.comment:comment
    List instances with their running state and user comment.
```

Options

```
      --all-projects   Display instances from all projects
  -c, --columns        Columns (default "ns46tSL")
      --fast           Fast mode (same as --columns=nsacPt)
  -f, --format         Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

`lxc manpage`

Generate manpages for all commands

Synopsis

Description: Generate manpages for all commands

```
lxc manpage <target> [flags]
```

Options

```
  -f, --format   Format (man|md|rest|yaml) (default "man")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

`lxc monitor`

Monitor a local or remote LXD server

Synopsis

Description: Monitor a local or remote LXD server

By default the monitor will listen to all message types.

```
lxc monitor [<remote>:] [flags]
```

Examples

```
  lxc monitor --type=logging
      Only show log messages.

  lxc monitor --pretty --type=logging --loglevel=info
      Show a pretty log of messages with info level or higher.

  lxc monitor --type=lifecycle
      Only show lifecycle events.
```

Options

```
      --all-projects   Show events from all projects
  -f, --format         Format (json|pretty|yaml) (default "yaml")
      --loglevel       Minimum level for log messages (only available when using
pretty format)
      --pretty         Pretty rendering (short for --format=pretty)
      --type           Event type to listen for
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

`lxc move`

Move instances within or in between LXD servers

Synopsis

Description: Move instances within or in between LXD servers

Transfer modes (–mode):

- pull: Target server pulls the data from the source server (source must listen on network)
- push: Source server pushes the data to the target server (target must listen on network)
- relay: The CLI connects to both source and server and proxies the data (both source and target must listen on network)

The pull transfer mode is the default as it is compatible with all LXD versions.

```
lxc move [<remote>:]<instance>[/<snapshot>] [<remote>:][<instance>[/<snapshot>]]
[flags]
```

Examples

```
  lxc move [<remote>:]<source instance> [<remote>:][<destination instance>] [--
instance-only]
      Move an instance between two hosts, renaming it if destination name differs.

  lxc move <old name> <new name> [--instance-only]
      Rename a local instance.

  lxc move <instance>/<old snapshot name> <instance>/<new snapshot name>
      Rename a snapshot.
```

Options

```
      --allow-inconsistent   Ignore copy errors for volatile files
  -c, --config               Config key/value to apply to the target instance
  -d, --device               New key/value to apply to a specific device
      --instance-only        Move the instance without its snapshots
      --mode                 Transfer mode. One of pull, push or relay. (default
"pull")
      --no-profiles          Unset all profiles on the target instance
  -p, --profile              Profile to apply to the target instance
      --stateless            Copy a stateful instance as stateless
  -s, --storage              Storage pool name
      --target               Cluster member name
      --target-project       Copy to a project different from the source
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
```

(continues on next page)

(continued from previous page)

```
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

lxc network

Manage and attach instances to networks

Synopsis

Description: Manage and attach instances to networks

```
lxc network [flags]
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD
- *lxc network acl* (page 791) - Manage network ACLs
- *lxc network attach* (page 799) - Attach network interfaces to instances
- *lxc network attach-profile* (page 800) - Attach network interfaces to profiles
- *lxc network create* (page 800) - Create new networks
- *lxc network delete* (page 801) - Delete networks
- *lxc network detach* (page 802) - Detach network interfaces from instances
- *lxc network detach-profile* (page 802) - Detach network interfaces from profiles
- *lxc network edit* (page 803) - Edit network configurations as YAML
- *lxc network forward* (page 803) - Manage network forwards
- *lxc network get* (page 811) - Get values for network configuration keys

`lxc network acl`

Manage network ACLs

Synopsis

Description: Manage network ACLs

```
lxc network acl [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network acl rule* (page 795) - Manage network ACL rules
- *lxc network acl set* (page 797) - Set network ACL configuration keys
- *lxc network acl show* (page 798) - Show network ACL configurations
- *lxc network acl show-log* (page 798) - Show network ACL log
- *lxc network acl unset* (page 799) - Unset network ACL configuration keys

`lxc network acl create`

Create new network ACLs

Synopsis

Description: Create new network ACLs

```
lxc network acl create [<remote>:]<ACL> [key=value...] [flags]
```

Examples

```
  lxc network acl create a1

  lxc network acl create a1 < config.yaml
      Create network acl with configuration from config.yaml
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network acl* (page 791) - Manage network ACLs

`lxc network acl delete`

Delete network ACLs

Synopsis

Description: Delete network ACLs

```
lxc network acl delete [<remote>:]<ACL> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network acl* (page 791) - Manage network ACLs

`lxc network acl edit`

Edit network ACL configurations as YAML

Synopsis

Description: Edit network ACL configurations as YAML

```
lxc network acl edit [<remote>:]<ACL> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network acl* (page 791) - Manage network ACLs

`lxc network acl get`

Get values for network ACL configuration keys

Synopsis

Description: Get values for network ACL configuration keys

```
lxc network acl get [<remote>:]<ACL> <key> [flags]
```

Options

```
  -p, --property   Get the key as a network ACL property
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network acl* (page 791) - Manage network ACLs

`lxc network acl list`

List available network ACLS

Synopsis

Description: List available network ACL

```
lxc network acl list [<remote>:] [flags]
```

Options

```
      --all-projects   Display network ACLs from all projects
  -f, --format         Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network acl* (page 791) - Manage network ACLs

`lxc network acl rename`

Rename network ACLs

Synopsis

Description: Rename network ACLs

```
lxc network acl rename [<remote>:]<ACL> <new-name> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network acl* (page 791) - Manage network ACLs

`lxc network acl rule`

Manage network ACL rules

Synopsis

Description: Manage network ACL rules

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network acl* (page 791) - Manage network ACLs
- *lxc network acl rule add* (page 796) - Add rules to an ACL
- *lxc network acl rule remove* (page 796) - Remove rules from an ACL

`lxc network acl rule add`

Add rules to an ACL

Synopsis

Description: Add rules to an ACL

```
lxc network acl rule add [<remote>:]<ACL> <direction> <key>=<value>... [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network acl rule* (page 795) - Manage network ACL rules

`lxc network acl rule remove`

Remove rules from an ACL

Synopsis

Description: Remove rules from an ACL

```
lxc network acl rule remove [<remote>:]<ACL> <direction> <key>=<value>... [flags]
```

Options

```
      --force   Remove all rules that match
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network acl rule* (page 795) - Manage network ACL rules

`lxc network acl set`

Set network ACL configuration keys

Synopsis

Description: Set network ACL configuration keys

For backward compatibility, a single configuration key may still be set with: lxc network set [:]

```
lxc network acl set [<remote>:]<ACL> <key>=<value>... [flags]
```

Options

```
  -p, --property   Set the key as a network ACL property
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network acl* (page 791) - Manage network ACLs

`lxc network acl show`

Show network ACL configurations

Synopsis

Description: Show network ACL configurations

```
lxc network acl show [<remote>:]<ACL> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network acl* (page 791) - Manage network ACLs

`lxc network acl show-log`

Show network ACL log

Synopsis

Description: Show network ACL log

```
lxc network acl show-log [<remote>:]<ACL> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network acl* (page 791) - Manage network ACLs

lxc network acl unset

Unset network ACL configuration keys

Synopsis

Description: Unset network ACL configuration keys

```
lxc network acl unset [<remote>:]<ACL> <key> [flags]
```

Options

```
  -p, --property   Unset the key as a network ACL property
```

Options inherited from parent commands

```
      --debug            Show all debug messages
      --force-local      Force using the local unix socket
  -h, --help             Print help
      --project          Override the source project
  -q, --quiet            Don't show progress information
      --sub-commands     Use with help or --help to view sub-commands
  -v, --verbose          Show all information messages
      --version          Print version number
```

SEE ALSO

- *lxc network acl* (page 791) - Manage network ACLs

lxc network attach

Attach network interfaces to instances

Synopsis

Description: Attach new network interfaces to instances

```
lxc network attach [<remote>:]<network> <instance> [<device name>] [<interface
name>] [flags]
```

Options inherited from parent commands

```
      --debug            Show all debug messages
      --force-local      Force using the local unix socket
  -h, --help             Print help
```

(continues on next page)

```
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network* (page 790) - Manage and attach instances to networks

`lxc network attach-profile`

Attach network interfaces to profiles

Synopsis

Description: Attach network interfaces to profiles

```
lxc network attach-profile [<remote>:]<network> <profile> [<device name>] [
<interface name>] [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network* (page 790) - Manage and attach instances to networks

`lxc network create`

Create new networks

Synopsis

Description: Create new networks

```
lxc network create [<remote>:]<network> [key=value...] [flags]
```

Examples

```
lxc network create foo
    Create a new network called foo

lxc network create bar network=baz --type ovn
    Create a new OVN network called bar using baz as its uplink network
```

Options

```
    --target   Cluster member name
-t, --type     Network type
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc network* (page 790) - Manage and attach instances to networks

`lxc network delete`

Delete networks

Synopsis

Description: Delete networks

```
lxc network delete [<remote>:]<network> [flags]
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc network* (page 790) - Manage and attach instances to networks

`lxc network detach`

Detach network interfaces from instances

Synopsis

Description: Detach network interfaces from instances

```
lxc network detach [<remote>:]<network> <instance> [<device name>] [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network* (page 790) - Manage and attach instances to networks

`lxc network detach-profile`

Detach network interfaces from profiles

Synopsis

Description: Detach network interfaces from profiles

```
lxc network detach-profile [<remote>:]<network> <profile> [<device name>] [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network* (page 790) - Manage and attach instances to networks

`lxc network edit`

Edit network configurations as YAML

Synopsis

Description: Edit network configurations as YAML

```
lxc network edit [<remote>:]<network> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network* (page 790) - Manage and attach instances to networks

`lxc network forward`

Manage network forwards

Synopsis

Description: Manage network forwards

```
lxc network forward [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network* (page 790) - Manage and attach instances to networks
- *lxc network forward create* (page 804) - Create new network forwards
- *lxc network forward delete* (page 805) - Delete network forwards
- *lxc network forward edit* (page 805) - Edit network forward configurations as YAML
- *lxc network forward get* (page 806) - Get values for network forward configuration keys
- *lxc network forward list* (page 807) - List available network forwards
- *lxc network forward port* (page 807) - Manage network forward ports
- *lxc network forward set* (page 809) - Set network forward keys
- *lxc network forward show* (page 810) - Show network forward configurations
- *lxc network forward unset* (page 810) - Unset network forward configuration keys

`lxc network forward create`

Create new network forwards

Synopsis

Description: Create new network forwards

```
lxc network forward create [<remote>:]<network> [<listen_address>] [key=value...]
[flags]
```

Examples

```
  lxc network forward create n1 127.0.0.1

  lxc network forward create n1 127.0.0.1 < config.yaml
      Create a new network forward for network n1 from config.yaml
```

Options

```
      --allocate   Auto-allocate an IPv4 or IPv6 listen address. One of 'ipv4',
'ipv6'.
      --target     Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
```

(continues on next page)

```
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network forward* (page 803) - Manage network forwards

`lxc network forward delete`

Delete network forwards

Synopsis

Description: Delete network forwards

```
lxc network forward delete [<remote>:]<network> <listen_address> [flags]
```

Options

```
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network forward* (page 803) - Manage network forwards

`lxc network forward edit`

Edit network forward configurations as YAML

Synopsis

Description: Edit network forward configurations as YAML

```
lxc network forward edit [<remote>:]<network> <listen_address> [flags]
```

Options

```
    --target   Cluster member name
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc network forward* (page 803) - Manage network forwards

`lxc network forward get`

Get values for network forward configuration keys

Synopsis

Description: Get values for network forward configuration keys

```
lxc network forward get [<remote>:]<network> <listen_address> <key> [flags]
```

Options

```
-p, --property   Get the key as a network forward property
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc network forward* (page 803) - Manage network forwards

`lxc network forward list`

List available network forwards

Synopsis

Description: List available network forwards

```
lxc network forward list [<remote>:]<network> [flags]
```

Options

```
  -f, --format   Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network forward* (page 803) - Manage network forwards

`lxc network forward port`

Manage network forward ports

Synopsis

Description: Manage network forward ports

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network forward* (page 803) - Manage network forwards
- *lxc network forward port add* (page 808) - Add ports to a forward
- *lxc network forward port remove* (page 808) - Remove ports from a forward

`lxc network forward port add`

Add ports to a forward

Synopsis

Description: Add ports to a forward

```
lxc network forward port add [<remote>:]<network> <listen_address> <protocol>
<listen_port(s)> <target_address> [<target_port(s)>] [flags]
```

Options

```
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network forward port* (page 807) - Manage network forward ports

`lxc network forward port remove`

Remove ports from a forward

Synopsis

Description: Remove ports from a forward

```
lxc network forward port remove [<remote>:]<network> <listen_address> [<protocol>]
[<listen_port(s)>] [flags]
```

Options

```
    --force    Remove all ports that match
    --target   Cluster member name
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc network forward port* (page 807) - Manage network forward ports

`lxc network forward set`

Set network forward keys

Synopsis

Description: Set network forward keys

For backward compatibility, a single configuration key may still be set with: lxc network set [:] <listen_address>

```
lxc network forward set [<remote>:]<network> <listen_address> <key>=<value>...
[flags]
```

Options

```
-p, --property   Set the key as a network forward property
    --target     Cluster member name
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc network forward* (page 803) - Manage network forwards

`lxc network forward show`

Show network forward configurations

Synopsis

Description: Show network forward configurations

```
lxc network forward show [<remote>:]<network> <listen_address> [flags]
```

Options

```
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network forward* (page 803) - Manage network forwards

`lxc network forward unset`

Unset network forward configuration keys

Synopsis

Description: Unset network forward keys

```
lxc network forward unset [<remote>:]<network> <listen_address> <key> [flags]
```

Options

```
  -p, --property   Unset the key as a network forward property
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network forward* (page 803) - Manage network forwards

`lxc network get`

Get values for network configuration keys

Synopsis

Description: Get values for network configuration keys

```
lxc network get [<remote>:]<network> <key> [flags]
```

Options

```
  -p, --property   Get the key as a network property
      --target     Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network* (page 790) - Manage and attach instances to networks

lxc network info

Get runtime information on networks

Synopsis

Description: Get runtime information on networks

```
lxc network info [<remote>:]<network> [flags]
```

Options

```
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network* (page 790) - Manage and attach instances to networks

lxc network list

List available networks

Synopsis

Description: List available networks

```
lxc network list [<remote>:] [flags]
```

Options

```
      --all-projects   Display networks from all projects
  -f, --format         Format (csv|json|table|yaml|compact) (default "table")
      --target         Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network* (page 790) - Manage and attach instances to networks

lxc network list-allocations

List network allocations in use

Synopsis

Description: List network allocations in use

```
lxc network list-allocations [flags]
```

Options

```
      --all-projects     Run against all projects
  -f, --format           Format (csv|json|table|yaml|compact) (default "table")
  -p, --project string   Run again a specific project (default "default")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network* (page 790) - Manage and attach instances to networks

lxc network list-leases

List DHCP leases

Synopsis

Description: List DHCP leases

```
lxc network list-leases [<remote>:]<network> [flags]
```

Options

```
  -f, --format   Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network* (page 790) - Manage and attach instances to networks

lxc network load-balancer

Manage network load balancers

Synopsis

Description: Manage network load balancers

```
lxc network load-balancer [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network* (page 790) - Manage and attach instances to networks
- *lxc network load-balancer backend* (page 815) - Manage network load balancer backends
- *lxc network load-balancer create* (page 817) - Create new network load balancers
- *lxc network load-balancer delete* (page 818) - Delete network load balancers
- *lxc network load-balancer edit* (page 818) - Edit network load balancer configurations as YAML
- *lxc network load-balancer get* (page 819) - Get values for network load balancer configuration keys
- *lxc network load-balancer list* (page 820) - List available network load balancers
- *lxc network load-balancer port* (page 820) - Manage network load balancer ports
- *lxc network load-balancer set* (page 822) - Set network load balancer keys
- *lxc network load-balancer show* (page 823) - Show network load balancer configurations
- *lxc network load-balancer unset* (page 823) - Unset network load balancer configuration keys

`lxc network load-balancer backend`

Manage network load balancer backends

Synopsis

Description: Manage network load balancer backends

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network load-balancer* (page 814) - Manage network load balancers
- *lxc network load-balancer backend add* (page 816) - Add backends to a load balancer
- *lxc network load-balancer backend remove* (page 816) - Remove backends from a load balancer

lxc network load-balancer backend add

Add backends to a load balancer

Synopsis

Description: Add backend to a load balancer

```
lxc network load-balancer backend add [<remote>:]<network> <listen_address>
<backend_name> <target_address> [<target_port(s)>] [flags]
```

Options

```
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc network load-balancer backend* (page 815) - Manage network load balancer backends

lxc network load-balancer backend remove

Remove backends from a load balancer

Synopsis

Description: Remove backend from a load balancer

```
lxc network load-balancer backend remove [<remote>:]<network> <listen_address>
<backend_name> [flags]
```

Options

```
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network load-balancer backend* (page 815) - Manage network load balancer backends

`lxc network load-balancer create`

Create new network load balancers

Synopsis

Description: Create new network load balancers

```
lxc network load-balancer create [<remote>:]<network> [<listen_address>]
[key=value...] [flags]
```

Examples

```
  lxc network load-balancer create n1 127.0.0.1

  lxc network load-balancer create n1 127.0.0.1 < config.yaml
      Create network load-balancer for network n1 with configuration from config.
yaml
```

Options

```
      --allocate   Auto-allocate an IPv4 or IPv6 listen address. One of 'ipv4',
'ipv6'.
      --target     Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
```

(continues on next page)

```
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc network load-balancer* (page 814) - Manage network load balancers

`lxc network load-balancer delete`

Delete network load balancers

Synopsis

Description: Delete network load balancers

```
lxc network load-balancer delete [<remote>:]<network> <listen_address> [flags]
```

Options

```
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc network load-balancer* (page 814) - Manage network load balancers

`lxc network load-balancer edit`

Edit network load balancer configurations as YAML

Synopsis

Description: Edit network load balancer configurations as YAML

```
lxc network load-balancer edit [<remote>:]<network> <listen_address> [flags]
```

Options

```
    --target   Cluster member name
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc network load-balancer* (page 814) - Manage network load balancers

`lxc network load-balancer get`

Get values for network load balancer configuration keys

Synopsis

Description: Get values for network load balancer configuration keys

```
lxc network load-balancer get [<remote>:]<network> <listen_address> <key> [flags]
```

Options

```
-p, --property   Get the key as a network load balancer property
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc network load-balancer* (page 814) - Manage network load balancers

`lxc network load-balancer list`

List available network load balancers

Synopsis

Description: List available network load balancers

```
lxc network load-balancer list [<remote>:]<network> [flags]
```

Options

```
  -f, --format   Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network load-balancer* (page 814) - Manage network load balancers

`lxc network load-balancer port`

Manage network load balancer ports

Synopsis

Description: Manage network load balancer ports

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network load-balancer* (page 814) - Manage network load balancers
- *lxc network load-balancer port add* (page 821) - Add ports to a load balancer
- *lxc network load-balancer port remove* (page 821) - Remove ports from a load balancer

`lxc network load-balancer port add`

Add ports to a load balancer

Synopsis

Description: Add ports to a load balancer

```
lxc network load-balancer port add [<remote>:]<network> <listen_address>
<protocol> <listen_port(s)> <backend_name>[,<backend_name>...] [flags]
```

Options

```
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network load-balancer port* (page 820) - Manage network load balancer ports

`lxc network load-balancer port remove`

Remove ports from a load balancer

Synopsis

Description: Remove ports from a load balancer

```
lxc network load-balancer port remove [<remote>:]<network> <listen_address> [
<protocol>] [<listen_port(s)>] [flags]
```

Options

```
      --force    Remove all ports that match
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network load-balancer port* (page 820) - Manage network load balancer ports

`lxc network load-balancer set`

Set network load balancer keys

Synopsis

Description: Set network load balancer keys

For backward compatibility, a single configuration key may still be set with: lxc network set [:] <listen_address>

```
lxc network load-balancer set [<remote>:]<network> <listen_address> <key>=<value>.
.. [flags]
```

Options

```
  -p, --property   Set the key as a network load balancer property
      --target     Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network load-balancer* (page 814) - Manage network load balancers

`lxc network load-balancer show`

Show network load balancer configurations

Synopsis

Description: Show network load balancer configurations

```
lxc network load-balancer show [<remote>:]<network> <listen_address> [flags]
```

Options

```
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network load-balancer* (page 814) - Manage network load balancers

`lxc network load-balancer unset`

Unset network load balancer configuration keys

Synopsis

Description: Unset network load balancer keys

```
lxc network load-balancer unset [<remote>:]<network> <listen_address> <key>
[flags]
```

Options

```
  -p, --property   Unset the key as a network load balancer property
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc network load-balancer* (page 814) - Manage network load balancers

`lxc network peer`

Manage network peerings

Synopsis

Description: Manage network peerings

```
lxc network peer [flags]
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc network* (page 790) - Manage and attach instances to networks
- *lxc network peer create* (page 825) - Create new network peering
- *lxc network peer delete* (page 825) - Delete network peerings
- *lxc network peer edit* (page 826) - Edit network peer configurations as YAML
- *lxc network peer get* (page 826) - Get values for network peer configuration keys
- *lxc network peer list* (page 827) - List available network peers
- *lxc network peer set* (page 827) - Set network peer keys
- *lxc network peer show* (page 828) - Show network peer configurations

- *lxc network peer unset* (page 829) - Unset network peer configuration keys

`lxc network peer create`

Create new network peering

Synopsis

Description: Create new network peering

```
lxc network peer create [<remote>:]<network> <peer_name> <[target project/]target_
network> [key=value...] [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network peer* (page 824) - Manage network peerings

`lxc network peer delete`

Delete network peerings

Synopsis

Description: Delete network peerings

```
lxc network peer delete [<remote>:]<network> <peer_name> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network peer* (page 824) - Manage network peerings

`lxc network peer edit`

Edit network peer configurations as YAML

Synopsis

Description: Edit network peer configurations as YAML

```
lxc network peer edit [<remote>:]<network> <peer_name> [flags]
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc network peer* (page 824) - Manage network peerings

`lxc network peer get`

Get values for network peer configuration keys

Synopsis

Description: Get values for network peer configuration keys

```
lxc network peer get [<remote>:]<network> <peer_name> <key> [flags]
```

Options

```
  -p, --property   Get the key as a network peer property
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
```

(continues on next page)

```
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc network peer* (page 824) - Manage network peerings

`lxc network peer list`

List available network peers

Synopsis

Description: List available network peers

```
lxc network peer list [<remote>:]<network> [flags]
```

Options

```
  -f, --format   Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc network peer* (page 824) - Manage network peerings

`lxc network peer set`

Set network peer keys

Synopsis

Description: Set network peer keys

For backward compatibility, a single configuration key may still be set with: lxc network set [:] <peer_name>

```
lxc network peer set [<remote>:]<network> <peer_name> <key>=<value>... [flags]
```

Options

```
  -p, --property   Set the key as a network peer property
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network peer* (page 824) - Manage network peerings

`lxc network peer show`

Show network peer configurations

Synopsis

Description: Show network peer configurations

```
lxc network peer show [<remote>:]<network> <peer name> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network peer* (page 824) - Manage network peerings `lxc network peer unset`

Unset network peer configuration keys

Synopsis

Description: Unset network peer keys

```
lxc network peer unset [<remote>:]<network> <peer_name> <key> [flags]
```

Options

```
  -p, --property   Unset the key as a network peer property
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network peer* (page 824) - Manage network peerings

`lxc network rename`

Rename networks

Synopsis

Description: Rename networks

```
lxc network rename [<remote>:]<network> <new-name> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network* (page 790) - Manage and attach instances to networks

`lxc network set`

Set network configuration keys

Synopsis

Description: Set network configuration keys

For backward compatibility, a single configuration key may still be set with: lxc network set [:]

```
lxc network set [<remote>:]<network> <key>=<value>... [flags]
```

Options

```
  -p, --property   Set the key as a network property
      --target     Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network* (page 790) - Manage and attach instances to networks

`lxc network show`

Show network configurations

Synopsis

Description: Show network configurations

```
lxc network show [<remote>:]<network> [flags]
```

Options

```
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network* (page 790) - Manage and attach instances to networks

`lxc network unset`

Unset network configuration keys

Synopsis

Description: Unset network configuration keys

```
lxc network unset [<remote>:]<network> <key> [flags]
```

Options

```
  -p, --property   Unset the key as a network property
      --target     Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network* (page 790) - Manage and attach instances to networks

lxc network zone

Manage network zones

Synopsis

Description: Manage network zones

```
lxc network zone [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network* (page 790) - Manage and attach instances to networks
- *lxc network zone create* (page 832) - Create new network zones
- *lxc network zone delete* (page 833) - Delete network zones
- *lxc network zone edit* (page 834) - Edit network zone configurations as YAML
- *lxc network zone get* (page 834) - Get values for network zone configuration keys
- *lxc network zone list* (page 835) - List available network zones
- *lxc network zone record* (page 835) - Manage network zone records
- *lxc network zone set* (page 843) - Set network zone configuration keys
- *lxc network zone show* (page 843) - Show network zone configurations
- *lxc network zone unset* (page 844) - Unset network zone configuration keys

lxc network zone create

Create new network zones

Synopsis

Description: Create new network zones

```
lxc network zone create [<remote>:]<Zone> [key=value...] [flags]
```

Examples

```
  lxc network zone create z1

  lxc network zone create z1 < config.yaml
      Create network zone z1 with configuration from config.yaml
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network zone* (page 832) - Manage network zones

`lxc network zone delete`

Delete network zones

Synopsis

Description: Delete network zones

```
lxc network zone delete [<remote>:]<Zone> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network zone* (page 832) - Manage network zones

`lxc network zone edit`

Edit network zone configurations as YAML

Synopsis

Description: Edit network zone configurations as YAML

```
lxc network zone edit [<remote>:]<Zone> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network zone* (page 832) - Manage network zones

`lxc network zone get`

Get values for network zone configuration keys

Synopsis

Description: Get values for network zone configuration keys

```
lxc network zone get [<remote>:]<Zone> <key> [flags]
```

Options

```
  -p, --property   Get the key as a network zone property
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
```

(continues on next page)

(continued from previous page)

```
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network zone* (page 832) - Manage network zones

lxc network zone list

List available network zones

Synopsis

Description: List available network zone

```
lxc network zone list [<remote>:] [flags]
```

Options

```
      --all-projects   Display network zones from all projects
  -f, --format         Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network zone* (page 832) - Manage network zones

lxc network zone record

Manage network zone records

Synopsis

Description: Manage network zone records

```
lxc network zone record [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network zone* (page 832) - Manage network zones
- *lxc network zone record create* (page 836) - Create new network zone record
- *lxc network zone record delete* (page 837) - Delete network zone record
- *lxc network zone record edit* (page 837) - Edit network zone record configurations as YAML
- *lxc network zone record entry* (page 838) - Manage network zone record entries
- *lxc network zone record get* (page 840) - Get values for network zone record configuration keys
- *lxc network zone record list* (page 840) - List available network zone records
- *lxc network zone record set* (page 841) - Set network zone record configuration keys
- *lxc network zone record show* (page 841) - Show network zone record configuration
- *lxc network zone record unset* (page 842) - Unset network zone record configuration keys

`lxc network zone record create`

Create new network zone record

Synopsis

Description: Create new network zone record

```
lxc network zone record create [<remote>:]<zone> <record> [key=value...] [flags]
```

Examples

```
  lxc network zone record create z1 r1

  lxc network zone record create z1 r1 < config.yaml
      Create record r1 for zone z1 with configuration from config.yaml
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network zone record* (page 835) - Manage network zone records

`lxc network zone record delete`

Delete network zone record

Synopsis

Description: Delete network zone record

```
lxc network zone record delete [<remote>:]<zone> <record> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network zone record* (page 835) - Manage network zone records

`lxc network zone record edit`

Edit network zone record configurations as YAML

Synopsis

Description: Edit network zone record configurations as YAML

```
lxc network zone record edit [<remote>:]<zone> <record> [flags]
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc network zone record* (page 835) - Manage network zone records

`lxc network zone record entry`

Manage network zone record entries

Synopsis

Description: Manage network zone record entries

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc network zone record* (page 835) - Manage network zone records
- *lxc network zone record entry add* (page 838) - Add a network zone record entry
- *lxc network zone record entry remove* (page 839) - Remove a network zone record entry

`lxc network zone record entry add`

Add a network zone record entry

Synopsis

Description: Add entries to a network zone record

```
lxc network zone record entry add [<remote>:]<zone> <record> <type> <value>
[flags]
```

Options

```
      --ttl   Entry TTL
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network zone record entry* (page 838) - Manage network zone record entries

`lxc network zone record entry remove`

Remove a network zone record entry

Synopsis

Description: Remove entries from a network zone record

```
lxc network zone record entry remove [<remote>:]<zone> <record> <type> <value>
[flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network zone record entry* (page 838) - Manage network zone record entries

`lxc network zone record get`

Get values for network zone record configuration keys

Synopsis

Description: Get values for network zone record configuration keys

```
lxc network zone record get [<remote>:]<zone> <record> <key> [flags]
```

Options

```
  -p, --property   Get the key as a network zone record property
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network zone record* (page 835) - Manage network zone records

`lxc network zone record list`

List available network zone records

Synopsis

Description: List available network zone records

```
lxc network zone record list [<remote>:]<zone> [flags]
```

Options

```
  -f, --format   Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc network zone record* (page 835) - Manage network zone records

`lxc network zone record set`

Set network zone record configuration keys

Synopsis

Description: Set network zone record configuration keys

```
lxc network zone record set [<remote>:]<zone> <record> <key>=<value>... [flags]
```

Options

```
  -p, --property   Set the key as a network zone record property
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc network zone record* (page 835) - Manage network zone records

`lxc network zone record show`

Show network zone record configuration

Synopsis

Description: Show network zone record configurations

```
lxc network zone record show [<remote>:]<zone> <record> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network zone record* (page 835) - Manage network zone records

`lxc network zone record unset`

Unset network zone record configuration keys

Synopsis

Description: Unset network zone record configuration keys

```
lxc network zone record unset [<remote>:]<zone> <record> <key> [flags]
```

Options

```
  -p, --property   Unset the key as a network zone record property
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network zone record* (page 835) - Manage network zone records

`lxc network zone set`

Set network zone configuration keys

Synopsis

Description: Set network zone configuration keys

For backward compatibility, a single configuration key may still be set with: lxc network set [:]

```
lxc network zone set [<remote>:]<Zone> <key>=<value>... [flags]
```

Options

```
  -p, --property   Set the key as a network zone property
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network zone* (page 832) - Manage network zones

`lxc network zone show`

Show network zone configurations

Synopsis

Description: Show network zone configurations

```
lxc network zone show [<remote>:]<Zone> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network zone* (page 832) - Manage network zones

`lxc network zone unset`

Unset network zone configuration keys

Synopsis

Description: Unset network zone configuration keys

```
lxc network zone unset [<remote>:]<Zone> <key> [flags]
```

Options

```
  -p, --property   Unset the key as a network zone property
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc network zone* (page 832) - Manage network zones

`lxc operation`

List, show and delete background operations

Synopsis

Description: List, show and delete background operations

```
lxc operation [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD
- *lxc operation delete* (page 845) - Delete a background operation (will attempt to cancel)
- *lxc operation list* (page 846) - List background operations
- *lxc operation show* (page 846) - Show details on a background operation

`lxc operation delete`

Delete a background operation (will attempt to cancel)

Synopsis

Description: Delete a background operation (will attempt to cancel)

```
lxc operation delete [<remote>:]<operation> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc operation* (page 844) - List, show and delete background operations

`lxc operation list`

List background operations

Synopsis

Description: List background operations

```
lxc operation list [<remote>:] [flags]
```

Options

```
      --all-projects   List operations from all projects
  -f, --format         Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc operation* (page 844) - List, show and delete background operations

`lxc operation show`

Show details on a background operation

Synopsis

Description: Show details on a background operation

```
lxc operation show [<remote>:]<operation> [flags]
```

Examples

```
  lxc operation show 344a79e4-d88a-45bf-9c39-c72c26f6ab8a
      Show details on that operation UUID
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc operation* (page 844) - List, show and delete background operations

`lxc pause`

Pause instances

Synopsis

Description: Pause instances

```
lxc pause [<remote>:]<instance> [[<remote>:]<instance>...] [flags]
```

Options

```
      --all    Run against all instances
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

`lxc profile`

Manage profiles

Synopsis

Description: Manage profiles

```
lxc profile [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD
- *lxc profile add* (page 848) - Add profiles to instances
- *lxc profile assign* (page 849) - Assign sets of profiles to instances
- *lxc profile copy* (page 850) - Copy profiles
- *lxc profile create* (page 850) - Create profiles
- *lxc profile delete* (page 851) - Delete profiles
- *lxc profile device* (page 852) - Manage devices
- *lxc profile edit* (page 856) - Edit profile configurations as YAML
- *lxc profile get* (page 857) - Get values for profile configuration keys
- *lxc profile list* (page 857) - List profiles
- *lxc profile remove* (page 858) - Remove profiles from instances
- *lxc profile rename* (page 859) - Rename profiles
- *lxc profile set* (page 859) - Set profile configuration keys
- *lxc profile show* (page 860) - Show profile configurations
- *lxc profile unset* (page 860) - Unset profile configuration keys

`lxc profile add`

Add profiles to instances

Synopsis

Description: Add profiles to instances

```
lxc profile add [<remote>:]<instance> <profile> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc profile* (page 847) - Manage profiles

`lxc profile assign`

Assign sets of profiles to instances

Synopsis

Description: Assign sets of profiles to instances

```
lxc profile assign [<remote>:]<instance> <profiles> [flags]
```

Examples

```
  lxc profile assign foo default,bar
      Set the profiles for "foo" to "default" and "bar".

  lxc profile assign foo default
      Reset "foo" to only using the "default" profile.

  lxc profile assign foo ''
      Remove all profile from "foo"
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
```

(continues on next page)

```
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc profile* (page 847) - Manage profiles

`lxc profile copy`

Copy profiles

Synopsis

Description: Copy profiles

```
lxc profile copy [<remote>:]<profile> [<remote>:]<profile> [flags]
```

Options

```
      --refresh          Update the target profile from the source if it already
exists
      --target-project   Copy to a project different from the source
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc profile* (page 847) - Manage profiles

`lxc profile create`

Create profiles

Synopsis

Description: Create profiles

```
lxc profile create [<remote>:]<profile> [flags]
```

Examples

```
lxc profile create p1

lxc profile create p1 < config.yaml
    Create profile with configuration from config.yaml
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc profile* (page 847) - Manage profiles

`lxc profile delete`

Delete profiles

Synopsis

Description: Delete profiles

```
lxc profile delete [<remote>:]<profile> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc profile* (page 847) - Manage profiles

`lxc profile device`

Manage devices

Synopsis

Description: Manage devices

```
lxc profile device [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc profile* (page 847) - Manage profiles
- *lxc profile device add* (page 852) - Add instance devices
- *lxc profile device get* (page 853) - Get values for device configuration keys
- *lxc profile device list* (page 854) - List instance devices
- *lxc profile device remove* (page 854) - Remove instance devices
- *lxc profile device set* (page 855) - Set device configuration keys
- *lxc profile device show* (page 855) - Show full device configuration
- *lxc profile device unset* (page 856) - Unset device configuration keys

`lxc profile device add`

Add instance devices

Synopsis

Description: Add instance devices

```
lxc profile device add [<remote>:]<profile> <device> <type> [key=value...] [flags]
```

Examples

```
  lxc profile device add [<remote>:]profile1 <device-name> disk source=/share/c1
path=/opt
      Will mount the host's /share/c1 onto /opt in the instance.

  lxc profile device add [<remote>:]profile1 <device-name> disk pool=some-pool
source=some-volume path=/opt
      Will mount the some-volume volume on some-pool onto /opt in the instance.
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc profile device* (page 852) - Manage devices

`lxc profile device get`

Get values for device configuration keys

Synopsis

Description: Get values for device configuration keys

```
lxc profile device get [<remote>:]<profile> <device> <key> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc profile device* (page 852) - Manage devices

`lxc profile device list`

List instance devices

Synopsis

Description: List instance devices

```
lxc profile device list [<remote>:]<profile> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc profile device* (page 852) - Manage devices

`lxc profile device remove`

Remove instance devices

Synopsis

Description: Remove instance devices

```
lxc profile device remove [<remote>:]<profile> <name>... [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc profile device* (page 852) - Manage devices

`lxc profile device set`

Set device configuration keys

Synopsis

Description: Set device configuration keys

For backward compatibility, a single configuration key may still be set with: lxc profile device set [:]

```
lxc profile device set [<remote>:]<profile> <device> <key>=<value>... [flags]
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc profile device* (page 852) - Manage devices

`lxc profile device show`

Show full device configuration

Synopsis

Description: Show full device configuration

```
lxc profile device show [<remote>:]<profile> [flags]
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
```

(continues on next page)

(continued from previous page)

```
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc profile device* (page 852) - Manage devices

`lxc profile device unset`

Unset device configuration keys

Synopsis

Description: Unset device configuration keys

```
lxc profile device unset [<remote>:]<profile> <device> <key> [flags]
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc profile device* (page 852) - Manage devices

`lxc profile edit`

Edit profile configurations as YAML

Synopsis

Description: Edit profile configurations as YAML

```
lxc profile edit [<remote>:]<profile> [flags]
```

Examples

```
  lxc profile edit <profile> < profile.yaml
      Update a profile using the content of profile.yaml
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc profile* (page 847) - Manage profiles

`lxc profile get`

Get values for profile configuration keys

Synopsis

Description: Get values for profile configuration keys

```
lxc profile get [<remote>:]<profile> <key> [flags]
```

Options

```
  -p, --property   Get the key as a profile property
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc profile* (page 847) - Manage profiles

`lxc profile list`

List profiles

Synopsis

Description: List profiles

The -c option takes a (optionally comma-separated) list of arguments that control which profile attributes to output when displaying in table or csv format.

Default column layout is: ndu

Column shorthand chars: n - Profile Name d - Description u - Used By

```
lxc profile list [<remote>:] [flags]
```

Options

```
      --all-projects   Display profiles from all projects
  -c, --columns        Columns (default "ndu")
  -f, --format         Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc profile* (page 847) - Manage profiles

`lxc profile remove`

Remove profiles from instances

Synopsis

Description: Remove profiles from instances

```
lxc profile remove [<remote>:]<instance> <profile> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
```

(continues on next page)

(continued from previous page)

```
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc profile* (page 847) - Manage profiles

`lxc profile rename`

Rename profiles

Synopsis

Description: Rename profiles

```
lxc profile rename [<remote>:]<profile> <new-name> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc profile* (page 847) - Manage profiles

`lxc profile set`

Set profile configuration keys

Synopsis

Description: Set profile configuration keys

For backward compatibility, a single configuration key may still be set with: lxc profile set [:]

```
lxc profile set [<remote>:]<profile> <key>=<value>... [flags]
```

Options

```
-p, --property   Set the key as a profile property
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc profile* (page 847) - Manage profiles

`lxc profile show`

Show profile configurations

Synopsis

Description: Show profile configurations

```
lxc profile show [<remote>:]<profile> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc profile* (page 847) - Manage profiles

`lxc profile unset`

Unset profile configuration keys

Synopsis

Description: Unset profile configuration keys

```
lxc profile unset [<remote>:]<profile> <key> [flags]
```

Options

```
  -p, --property   Unset the key as a profile property
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc profile* (page 847) - Manage profiles

`lxc project`

Manage projects

Synopsis

Description: Manage projects

```
lxc project [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD
- *lxc project create* (page 862) - Create projects
- *lxc project delete* (page 863) - Delete projects
- *lxc project edit* (page 863) - Edit project configurations as YAML
- *lxc project get* (page 864) - Get values for project configuration keys
- *lxc project info* (page 865) - Get a summary of resource allocations
- *lxc project list* (page 865) - List projects
- *lxc project rename* (page 866) - Rename projects
- *lxc project set* (page 866) - Set project configuration keys
- *lxc project show* (page 867) - Show project options
- *lxc project switch* (page 868) - Switch the current project
- *lxc project unset* (page 868) - Unset project configuration keys

`lxc project create`

Create projects

Synopsis

Description: Create projects

```
lxc project create [<remote>:]<project> [flags]
```

Examples

```
  lxc project create p1

  lxc project create p1 < config.yaml
      Create a project with configuration from config.yaml
```

Options

```
  -c, --config    Config key/value to apply to the new project
  -n, --network   Add a NIC device to the default profile connected to the
specified network
  -s, --storage   Add a storage pool to be used as the root device in the default
profile
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc project* (page 861) - Manage projects

`lxc project delete`

Delete projects

Synopsis

Description: Delete projects

```
lxc project delete [<remote>:]<project> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc project* (page 861) - Manage projects

`lxc project edit`

Edit project configurations as YAML

Synopsis

Description: Edit project configurations as YAML

```
lxc project edit [<remote>:]<project> [flags]
```

Examples

```
lxc project edit <project> < project.yaml
    Update a project using the content of project.yaml
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc project* (page 861) - Manage projects

`lxc project get`

Get values for project configuration keys

Synopsis

Description: Get values for project configuration keys

```
lxc project get [<remote>:]<project> <key> [flags]
```

Options

```
-p, --property   Get the key as a project property
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc project* (page 861) - Manage projects

`lxc project info`

Get a summary of resource allocations

Synopsis

Description: Get a summary of resource allocations

```
lxc project info [<remote>:]<project> [flags]
```

Options

```
  -f, --format   Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc project* (page 861) - Manage projects

`lxc project list`

List projects

Synopsis

Description: List projects

```
lxc project list [<remote>:] [flags]
```

Options

```
  -f, --format   Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc project* (page 861) - Manage projects

`lxc project rename`

Rename projects

Synopsis

Description: Rename projects

```
lxc project rename [<remote>:]<project> <new-name> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc project* (page 861) - Manage projects

`lxc project set`

Set project configuration keys

Synopsis

Description: Set project configuration keys

For backward compatibility, a single configuration key may still be set with: lxc project set [:]

```
lxc project set [<remote>:]<project> <key>=<value>... [flags]
```

Options

```
  -p, --property   Set the key as a project property
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc project* (page 861) - Manage projects

`lxc project show`

Show project options

Synopsis

Description: Show project options

```
lxc project show [<remote>:]<project> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc project* (page 861) - Manage projects

lxc project switch

Switch the current project

Synopsis

Description: Switch the current project

```
lxc project switch [<remote>:]<project> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc project* (page 861) - Manage projects

lxc project unset

Unset project configuration keys

Synopsis

Description: Unset project configuration keys

```
lxc project unset [<remote>:]<project> <key> [flags]
```

Options

```
  -p, --property   Unset the key as a project property
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc project* (page 861) - Manage projects

`lxc publish`

Publish instances as images

Synopsis

Description: Publish instances as images

```
lxc publish [<remote>:]<instance>[/<snapshot>] [<remote>:] [flags] [key=value...]
```

Options

```
      --alias              New alias to define at target
      --compression none   Compression algorithm to use (none for uncompressed)
      --expire             Image expiration date (format: rfc3339)
  -f, --force              Stop the instance if currently running
      --public             Make the image public (accessible to unauthenticated
clients as well)
      --reuse              If the image alias already exists, delete and create a
new one
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

`lxc query`

Send a raw query to LXD

Synopsis

Description: Send a raw query to LXD

```
lxc query [<remote>:]<API path> [flags]
```

Examples

```
lxc query -X DELETE --wait /1.0/instances/c1
    Delete local instance "c1".
```

Options

```
-d, --data      Input data
    --raw       Print the raw response
-X, --request   Action (default "GET")
    --wait      Wait for the operation to complete
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

`lxc rebuild`

Rebuild instances

Synopsis

Description: Wipe the instance root disk and re-initialize. The original image is used to re-initialize the instance if a different image or –empty is not specified.

```
lxc rebuild [<remote>:]<image> [<remote>:]<instance> [flags]
```

Options

```
    --empty   Rebuild as an empty instance
-f, --force   If an instance is running, stop it and then rebuild it
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
```

(continues on next page)

(continued from previous page)

```
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

`lxc remote`

Manage the list of remote servers

Synopsis

Description: Manage the list of remote servers

```
lxc remote [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD
- *lxc remote add* (page 872) - Add new remote servers
- *lxc remote get-default* (page 872) - Show the default remote
- *lxc remote list* (page 873) - List the available remotes
- *lxc remote remove* (page 873) - Remove remotes
- *lxc remote rename* (page 874) - Rename remotes
- *lxc remote set-url* (page 875) - Set the URL for the remote
- *lxc remote switch* (page 875) - Switch the default remote

`lxc remote add`

Add new remote servers

Synopsis

Description: Add new remote servers

URL for remote resources must be HTTPS (https://).

Basic authentication can be used when combined with the "simplestreams" protocol: lxc remote add some-name https://LOGIN:PASSWORD@example.com/some/path –protocol=simplestreams

```
lxc remote add [<remote>] <IP|FQDN|URL|token> [flags]
```

Options

```
      --accept-certificate   Accept certificate
      --auth-type            Server authentication type (tls or oidc)
      --password             Remote admin password
      --project              Project to use for the remote
      --protocol             Server protocol (lxd or simplestreams)
      --public               Public image server
      --token                Remote trust token
```

Options inherited from parent commands

```
      --debug         Show all debug messages
      --force-local   Force using the local unix socket
  -h, --help          Print help
  -q, --quiet         Don't show progress information
      --sub-commands  Use with help or --help to view sub-commands
  -v, --verbose       Show all information messages
      --version       Print version number
```

SEE ALSO

- *lxc remote* (page 871) - Manage the list of remote servers

`lxc remote get-default`

Show the default remote

Synopsis

Description: Show the default remote

```
lxc remote get-default [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc remote* (page 871) - Manage the list of remote servers

`lxc remote list`

List the available remotes

Synopsis

Description: List the available remotes

```
lxc remote list [flags]
```

Options

```
  -f, --format   Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc remote* (page 871) - Manage the list of remote servers

`lxc remote remove`

Remove remotes

Synopsis

Description: Remove remotes

```
lxc remote remove <remote> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc remote* (page 871) - Manage the list of remote servers

`lxc remote rename`

Rename remotes

Synopsis

Description: Rename remotes

```
lxc remote rename <remote> <new-name> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc remote* (page 871) - Manage the list of remote servers

lxc remote set-url

Set the URL for the remote

Synopsis

Description: Set the URL for the remote

```
lxc remote set-url <remote> <URL> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc remote* (page 871) - Manage the list of remote servers

lxc remote switch

Switch the default remote

Synopsis

Description: Switch the default remote

```
lxc remote switch <remote> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc remote* (page 871) - Manage the list of remote servers

lxc rename

Rename instances and snapshots

Synopsis

Description: Rename instances and snapshots

```
lxc rename [<remote>:]<instance>[/<snapshot>] <instance>[/<snapshot>] [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

lxc restart

Restart instances

Synopsis

Description: Restart instances

The opposite of "lxc pause" is "lxc start".

```
lxc restart [<remote>:]<instance> [[<remote>:]<instance>...] [flags]
```

Options

```
      --all                    Run against all instances
      --console[="console"]    Immediately attach to the console
  -f, --force                  Force the instance to stop
      --timeout                Time to wait for the instance to shutdown cleanly
(default -1)
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

`lxc restore`

Restore instances from snapshots

Synopsis

Description: Restore instances from snapshots

If –stateful is passed, then the running state will be restored too.

```
lxc restore [<remote>:]<instance> <snapshot> [flags]
```

Examples

```
  lxc snapshot u1 snap0
      Create the snapshot.

  lxc restore u1 snap0
      Restore the snapshot.
```

Options

```
      --stateful   Whether or not to restore the instance's running state from
snapshot (if available)
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

`lxc snapshot`

Create instance snapshots

Synopsis

Description: Create instance snapshots

When –stateful is used, LXD attempts to checkpoint the instance's running state, including process memory state, TCP connections, …

```
lxc snapshot [<remote>:]<instance> [<snapshot name>] [flags]
```

Examples

```
  lxc snapshot u1 snap0
          Create a snapshot of "u1" called "snap0".

          lxc snapshot u1 snap0 < config.yaml
                  Create a snapshot of "u1" called "snap0" with the configuration
from "config.yaml".
```

Options

```
      --no-expiry   Ignore any configured auto-expiry for the instance
      --reuse       If the snapshot name already exists, delete and create a new
one
      --stateful    Whether or not to snapshot the instance's running state
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

`lxc start`

Start instances

Synopsis

Description: Start instances

```
lxc start [<remote>:]<instance> [[<remote>:]<instance>...] [flags]
```

Options

```
      --all                     Run against all instances
      --console[="console"]     Immediately attach to the console
      --stateless               Ignore the instance state
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

`lxc stop`

Stop instances

Synopsis

Description: Stop instances

```
lxc stop [<remote>:]<instance> [[<remote>:]<instance>...] [flags]
```

Options

```
      --all                     Run against all instances
      --console[="console"]     Immediately attach to the console
  -f, --force                   Force the instance to stop
      --stateful                Store the instance state
      --timeout                 Time to wait for the instance to shutdown cleanly
(default -1)
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

`lxc storage`

Manage storage pools and volumes

Synopsis

Description: Manage storage pools and volumes

```
lxc storage [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD
- *lxc storage bucket* (page 881) - Manage storage buckets
- *lxc storage create* (page 891) - Create storage pools
- *lxc storage delete* (page 892) - Delete storage pools
- *lxc storage edit* (page 893) - Edit storage pool configurations as YAML
- *lxc storage get* (page 893) - Get values for storage pool configuration keys
- *lxc storage info* (page 894) - Show useful information about storage pools
- *lxc storage list* (page 895) - List available storage pools

- *lxc storage set* (page 895) - Set storage pool configuration keys
- *lxc storage show* (page 896) - Show storage pool configurations and resources
- *lxc storage unset* (page 897) - Unset storage pool configuration keys
- *lxc storage volume* (page 897) - Manage storage volumes

`lxc storage bucket`

Manage storage buckets

Synopsis

Description: Manage storage buckets.

```
lxc storage bucket [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage* (page 880) - Manage storage pools and volumes
- *lxc storage bucket create* (page 881) - Create new custom storage buckets
- *lxc storage bucket delete* (page 882) - Delete storage buckets
- *lxc storage bucket edit* (page 883) - Edit storage bucket configurations as YAML
- *lxc storage bucket get* (page 884) - Get values for storage bucket configuration keys
- *lxc storage bucket key* (page 884) - Manage storage bucket keys
- *lxc storage bucket list* (page 889) - List storage buckets
- *lxc storage bucket set* (page 889) - Set storage bucket configuration keys
- *lxc storage bucket show* (page 890) - Show storage bucket configurations
- *lxc storage bucket unset* (page 891) - Unset storage bucket configuration keys

`lxc storage bucket create`

Create new custom storage buckets

Synopsis

Description: Create new custom storage buckets

```
lxc storage bucket create [<remote>:]<pool> <bucket> [key=value...] [flags]
```

Examples

```
  lxc storage bucket create p1 b01
        Create a new storage bucket name b01 in storage pool p1

  lxc storage bucket create p1 b01 < config.yaml
        Create a new storage bucket name b01 in storage pool p1 using the
content of config.yaml
```

Options

```
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage bucket* (page 881) - Manage storage buckets

`lxc storage bucket delete`

Delete storage buckets

Synopsis

Description: Delete storage buckets

```
lxc storage bucket delete [<remote>:]<pool> <bucket> [flags]
```

Options

```
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage bucket* (page 881) - Manage storage buckets

`lxc storage bucket edit`

Edit storage bucket configurations as YAML

Synopsis

Description: Edit storage bucket configurations as YAML

```
lxc storage bucket edit [<remote>:]<pool> <bucket> [flags]
```

Examples

```
  lxc storage bucket edit [<remote>:]<pool> <bucket> < bucket.yaml
      Update a storage bucket using the content of bucket.yaml.
```

Options

```
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage bucket* (page 881) - Manage storage buckets

`lxc storage bucket get`

Get values for storage bucket configuration keys

Synopsis

Description: Get values for storage bucket configuration keys

```
lxc storage bucket get [<remote>:]<pool> <bucket> <key> [flags]
```

Options

```
  -p, --property   Get the key as a storage bucket property
      --target     Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage bucket* (page 881) - Manage storage buckets

`lxc storage bucket key`

Manage storage bucket keys

Synopsis

Description: Manage storage bucket keys.

```
lxc storage bucket key [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
```

(continues on next page)

(continued from previous page)

```
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage bucket* (page 881) - Manage storage buckets
- *lxc storage bucket key create* (page 885) - Create key for a storage bucket
- *lxc storage bucket key delete* (page 886) - Delete key from a storage bucket
- *lxc storage bucket key edit* (page 886) - Edit storage bucket key as YAML
- *lxc storage bucket key list* (page 887) - List storage bucket keys
- *lxc storage bucket key show* (page 888) - Show storage bucket key configurations

`lxc storage bucket key create`

Create key for a storage bucket

Synopsis

Description: Create key for a storage bucket

```
lxc storage bucket key create [<remote>:]<pool> <bucket> <key> [flags]
```

Examples

```
  lxc storage bucket key create p1 b01 k1
          Create a key called k1 for the bucket b01 in the pool p1.

  lxc storage bucket key create p1 b01 k1 < config.yaml
          Create a key called k1 for the bucket b01 in the pool p1 using the
content of config.yaml.
```

Options

```
      --access-key   Access key (auto-generated if empty)
      --role         Role (admin or read-only) (default "read-only")
      --secret-key   Secret key (auto-generated if empty)
      --target       Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage bucket key* (page 884) - Manage storage bucket keys

`lxc storage bucket key delete`

Delete key from a storage bucket

Synopsis

Description: Delete key from a storage bucket

```
lxc storage bucket key delete [<remote>:]<pool> <bucket> <key> [flags]
```

Options

```
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage bucket key* (page 884) - Manage storage bucket keys

`lxc storage bucket key edit`

Edit storage bucket key as YAML

Synopsis

Description: Edit storage bucket key as YAML

```
lxc storage bucket key edit [<remote>:]<pool> <bucket> <key> [flags]
```

Examples

```
  lxc storage bucket edit [<remote>:]<pool> <bucket> <key> < key.yaml
      Update a storage bucket key using the content of key.yaml.
```

Options

```
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage bucket key* (page 884) - Manage storage bucket keys

`lxc storage bucket key list`

List storage bucket keys

Synopsis

Description: List storage bucket keys

```
lxc storage bucket key list [<remote>:]<pool> <bucket> [flags]
```

Options

```
  -f, --format   Format (csv|json|table|yaml|compact) (default "table")
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage bucket key* (page 884) - Manage storage bucket keys

`lxc storage bucket key show`

Show storage bucket key configurations

Synopsis

Description: Show storage bucket key configurations

```
lxc storage bucket key show [<remote>:]<pool> <bucket> <key> [flags]
```

Examples

```
  lxc storage bucket key show default data foo
      Will show the properties of a bucket key called "foo" for a bucket called
"data" in the "default" pool.
```

Options

```
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage bucket key* (page 884) - Manage storage bucket keys

`lxc storage bucket list`

List storage buckets

Synopsis

Description: List storage buckets

```
lxc storage bucket list [<remote>:]<pool> [flags]
```

Options

```
      --all-projects   Display storage pool buckets from all projects
  -f, --format         Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage bucket* (page 881) - Manage storage buckets

`lxc storage bucket set`

Set storage bucket configuration keys

Synopsis

Description: Set storage bucket configuration keys

For backward compatibility, a single configuration key may still be set with: lxc storage bucket set [:]

```
lxc storage bucket set [<remote>:]<pool> <bucket> <key>=<value>... [flags]
```

Options

```
-p, --property   Set the key as a storage bucket property
    --target     Cluster member name
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc storage bucket* (page 881) - Manage storage buckets

`lxc storage bucket show`

Show storage bucket configurations

Synopsis

Description: Show storage bucket configurations

```
lxc storage bucket show [<remote>:]<pool> <bucket> [flags]
```

Examples

```
lxc storage bucket show default data
    Will show the properties of a bucket called "data" in the "default" pool.
```

Options

```
    --target   Cluster member name
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
```

(continues on next page)

```
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc storage bucket* (page 881) - Manage storage buckets

`lxc storage bucket unset`

Unset storage bucket configuration keys

Synopsis

Description: Unset storage bucket configuration keys

```
lxc storage bucket unset [<remote>:]<pool> <bucket> <key> [flags]
```

Options

```
  -p, --property   Unset the key as a storage bucket property
      --target     Cluster member name
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc storage bucket* (page 881) - Manage storage buckets

`lxc storage create`

Create storage pools

Synopsis

Description: Create storage pools

```
lxc storage create [<remote>:]<pool> <driver> [key=value...] [flags]
```

Examples

```
lxc storage create s1 dir

lxc storage create s1 dir < config.yaml
    Create a storage pool using the content of config.yaml.
```

Options

```
    --target   Cluster member name
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc storage* (page 880) - Manage storage pools and volumes

`lxc storage delete`

Delete storage pools

Synopsis

Description: Delete storage pools

```
lxc storage delete [<remote>:]<pool> [flags]
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc storage* (page 880) - Manage storage pools and volumes

`lxc storage edit`

Edit storage pool configurations as YAML

Synopsis

Description: Edit storage pool configurations as YAML

```
lxc storage edit [<remote>:]<pool> [flags]
```

Examples

```
  lxc storage edit [<remote>:]<pool> < pool.yaml
      Update a storage pool using the content of pool.yaml.
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc storage* (page 880) - Manage storage pools and volumes

`lxc storage get`

Get values for storage pool configuration keys

Synopsis

Description: Get values for storage pool configuration keys

```
lxc storage get [<remote>:]<pool> <key> [flags]
```

Options

```
  -p, --property   Get the key as a storage property
      --target     Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage* (page 880) - Manage storage pools and volumes

`lxc storage info`

Show useful information about storage pools

Synopsis

Description: Show useful information about storage pools

```
lxc storage info [<remote>:]<pool> [flags]
```

Options

```
      --bytes    Show the used and free space in bytes
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage* (page 880) - Manage storage pools and volumes

lxc storage list

List available storage pools

Synopsis

Description: List available storage pools

```
lxc storage list [<remote>:] [flags]
```

Options

```
  -f, --format   Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage* (page 880) - Manage storage pools and volumes

lxc storage set

Set storage pool configuration keys

Synopsis

Description: Set storage pool configuration keys

For backward compatibility, a single configuration key may still be set with: lxc storage set [:]

```
lxc storage set [<remote>:]<pool> <key> <value> [flags]
```

Options

```
  -p, --property   Set the key as a storage property
      --target     Cluster member name
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc storage* (page 880) - Manage storage pools and volumes

`lxc storage show`

Show storage pool configurations and resources

Synopsis

Description: Show storage pool configurations and resources

```
lxc storage show [<remote>:]<pool> [flags]
```

Options

```
      --resources   Show the resources available to the storage pool
      --target      Cluster member name
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc storage* (page 880) - Manage storage pools and volumes

lxc storage unset

Unset storage pool configuration keys

Synopsis

Description: Unset storage pool configuration keys

```
lxc storage unset [<remote>:]<pool> <key> [flags]
```

Options

```
  -p, --property   Unset the key as a storage property
      --target     Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage* (page 880) - Manage storage pools and volumes

lxc storage volume

Manage storage volumes

Synopsis

Description: Manage storage volumes

Unless specified through a prefix, all volume operations affect "custom" (user created) volumes.

```
lxc storage volume [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
```

(continues on next page)

(continued from previous page)

```
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc storage* (page 880) - Manage storage pools and volumes
- *lxc storage volume attach* (page 898) - Attach new storage volumes to instances
- *lxc storage volume attach-profile* (page 899) - Attach new storage volumes to profiles
- *lxc storage volume copy* (page 900) - Copy storage volumes
- *lxc storage volume create* (page 900) - Create new custom storage volumes
- *lxc storage volume delete* (page 901) - Delete storage volumes
- *lxc storage volume detach* (page 902) - Detach storage volumes from instances
- *lxc storage volume detach-profile* (page 902) - Detach storage volumes from profiles
- *lxc storage volume edit* (page 903) - Edit storage volume configurations as YAML
- *lxc storage volume export* (page 904) - Export custom storage volume
- *lxc storage volume get* (page 904) - Get values for storage volume configuration keys
- *lxc storage volume import* (page 905) - Import custom storage volumes
- *lxc storage volume info* (page 906) - Show storage volume state information
- *lxc storage volume list* (page 907) - List storage volumes
- *lxc storage volume move* (page 908) - Move storage volumes between pools
- *lxc storage volume rename* (page 908) - Rename storage volumes and storage volume snapshots
- *lxc storage volume restore* (page 909) - Restore storage volume snapshots
- *lxc storage volume set* (page 910) - Set storage volume configuration keys
- *lxc storage volume show* (page 911) - Show storage volume configurations
- *lxc storage volume snapshot* (page 912) - Snapshot storage volumes
- *lxc storage volume unset* (page 913) - Unset storage volume configuration keys

`lxc storage volume attach`

Attach new storage volumes to instances

Synopsis

Description: Attach new storage volumes to instances

must be one of “custom” or “virtual-machine”

```
lxc storage volume attach [<remote>:]<pool> [<type>/]<volume>[/<snapshot>]
<instance> [<device name>] [<path>] [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage volume* (page 897) - Manage storage volumes

`lxc storage volume attach-profile`

Attach new storage volumes to profiles

Synopsis

Description: Attach new storage volumes to profiles

must be one of "custom" or "virtual-machine"

```
lxc storage volume attach-profile [<remote:>]<pool> [<type>/]<volume>[/<snapshot>]
<profile> [<device name>] [<path>] [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage volume* (page 897) - Manage storage volumes

`lxc storage volume copy`

Copy storage volumes

Synopsis

Description: Copy storage volumes

```
lxc storage volume copy [<remote>:]<pool>/<volume>[/<snapshot>] [<remote>:]<pool>/
<volume> [flags]
```

Options

```
      --destination-target   Destination cluster member name
      --mode                 Transfer mode. One of pull (default), push or relay.
(default "pull")
      --refresh              Refresh and update the existing storage volume copies
      --target               Cluster member name
      --target-project       Copy to a project different from the source
      --volume-only          Copy the volume without its snapshots
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage volume* (page 897) - Manage storage volumes

`lxc storage volume create`

Create new custom storage volumes

Synopsis

Description: Create new custom storage volumes

```
lxc storage volume create [<remote>:]<pool> <volume> [key=value...] [flags]
```

Examples

```
lxc storage volume create p1 v1

lxc storage volume create p1 v1 < config.yaml
        Create storage volume v1 for pool p1 with configuration from config.
yaml.
```

Options

```
      --target   Cluster member name
      --type     Content type, block or filesystem (default "filesystem")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage volume* (page 897) - Manage storage volumes

`lxc storage volume delete`

Delete storage volumes

Synopsis

Description: Delete storage volumes

```
lxc storage volume delete [<remote>:]<pool> <volume>[/<snapshot>] [flags]
```

Options

```
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
```

(continues on next page)

(continued from previous page)

```
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage volume* (page 897) - Manage storage volumes

`lxc storage volume detach`

Detach storage volumes from instances

Synopsis

Description: Detach storage volumes from instances

```
lxc storage volume detach [<remote>:]<pool> [<type>/]<volume>[/<snapshot>]
<instance> [<device name>] [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage volume* (page 897) - Manage storage volumes

`lxc storage volume detach-profile`

Detach storage volumes from profiles

Synopsis

Description: Detach storage volumes from profiles

```
lxc storage volume detach-profile [<remote:>]<pool> [<type>/]<volume>[/<snapshot>]
<profile> [<device name>] [flags]
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc storage volume* (page 897) - Manage storage volumes

`lxc storage volume edit`

Edit storage volume configurations as YAML

Synopsis

Description: Edit storage volume configurations as YAML

```
lxc storage volume edit [<remote>:]<pool> [<type>/]<volume> [flags]
```

Examples

```
  Provide the type of the storage volume if it is not custom.
  Supported types are custom, image, container and virtual-machine.

  lxc storage volume edit [<remote>:]<pool> [<type>/]<volume> < volume.yaml
      Update a storage volume using the content of pool.yaml.
```

Options

```
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc storage volume* (page 897) - Manage storage volumes

`lxc storage volume export`

Export custom storage volume

Synopsis

Description: Export custom storage volume

```
lxc storage volume export [<remote>:]<pool> <volume> [<path>] [flags]
```

Options

```
      --compression         Define a compression algorithm: for backup or none
      --export-version      Use a different metadata format version than the
latest one supported by the server (to support imports on older LXD versions)
      --optimized-storage   Use storage driver optimized format (can only be
restored on a similar pool)
      --target              Cluster member name
      --volume-only         Export the volume without its snapshots
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage volume* (page 897) - Manage storage volumes

`lxc storage volume get`

Get values for storage volume configuration keys

Synopsis

Description: Get values for storage volume configuration keys

```
lxc storage volume get [<remote>:]<pool> [<type>/]<volume>[/<snapshot>] <key>
[flags]
```

Examples

```
  Provide the type of the storage volume if it is not custom.
  Supported types are custom, image, container and virtual-machine.

  Add the name of the snapshot if type is one of custom, container or virtual-
machine.

  lxc storage volume get default data size
      Returns the size of a custom volume "data" in pool "default".

  lxc storage volume get default virtual-machine/data snapshots.expiry
      Returns the snapshot expiration period for a virtual machine "data" in pool
"default".
```

Options

```
  -p, --property   Get the key as a storage volume property
      --target     Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage volume* (page 897) - Manage storage volumes

`lxc storage volume import`

Import custom storage volumes

Synopsis

Description: Import backups of custom volumes including their snapshots.

```
lxc storage volume import [<remote>:]<pool> <backup file> [<volume name>] [flags]
```

Examples

```
lxc storage volume import default backup0.tar.gz
                Create a new custom volume using backup0.tar.gz as the source.
```

Options

```
    --target   Cluster member name
    --type     Import type, backup or iso (default "backup")
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc storage volume* (page 897) - Manage storage volumes

`lxc storage volume info`

Show storage volume state information

Synopsis

Description: Show storage volume state information

```
lxc storage volume info [<remote>:]<pool> [<type>/]<volume> [flags]
```

Examples

```
Provide the type of the storage volume if it is not custom.
Supported types are custom, container and virtual-machine.

lxc storage volume info default data
    Returns state information for a custom volume "data" in pool "default".

lxc storage volume info default virtual-machine/data
    Returns state information for a virtual machine "data" in pool "default".
```

Options

```
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage volume* (page 897) - Manage storage volumes

`lxc storage volume list`

List storage volumes

Synopsis

Description: List storage volumes

The -c option takes a (optionally comma-separated) list of arguments that control which image attributes to output when displaying in table or csv format.

Column shorthand chars: p - Storage pool name c - Content type (filesystem or block) d - Description e - Project name L - Location of the instance (e.g. its cluster member) n - Name t - Type of volume (custom, image, container or virtual-machine) u - Number of references (used by) U - Current disk usage

```
lxc storage volume list [<remote>:][<pool>] [<filter>...] [flags]
```

Options

```
      --all-projects   All projects
  -c, --columns        Columns (default "petndcuL")
  -f, --format         Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
```

(continues on next page)

```
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage volume* (page 897) - Manage storage volumes

`lxc storage volume move`

Move storage volumes between pools

Synopsis

Description: Move storage volumes between pools

```
lxc storage volume move [<remote>:]<pool>/<volume> [<remote>:]<pool>/<volume>
[flags]
```

Options

```
      --destination-target   Destination cluster member name
      --mode                 Transfer mode, one of pull (default), push or relay
(default "pull")
      --target               Cluster member name
      --target-project       Move to a project different from the source
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage volume* (page 897) - Manage storage volumes

`lxc storage volume rename`

Rename storage volumes and storage volume snapshots

Synopsis

Description: Rename storage volumes

```
lxc storage volume rename [<remote>:]<pool> <old name>[/<old snapshot name>] <new
name>[/<new snapshot name>] [flags]
```

Options

```
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc storage volume* (page 897) - Manage storage volumes

`lxc storage volume restore`

Restore storage volume snapshots

Synopsis

Description: Restore storage volume snapshots

```
lxc storage volume restore [<remote>:]<pool> <volume> <snapshot> [flags]
```

Options

```
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
```

(continues on next page)

(continued from previous page)

```
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage volume* (page 897) - Manage storage volumes

`lxc storage volume set`

Set storage volume configuration keys

Synopsis

Description: Set storage volume configuration keys

For backward compatibility, a single configuration key may still be set with: lxc storage volume set [:] [/]

```
lxc storage volume set [<remote>:]<pool> [<type>/]<volume> <key>=<value>...
[flags]
```

Examples

```
  Provide the type of the storage volume if it is not custom.
  Supported types are custom, image, container and virtual-machine.

  lxc storage volume set default data size=1GiB
      Sets the size of a custom volume "data" in pool "default" to 1 GiB.

  lxc storage volume set default virtual-machine/data snapshots.expiry=7d
      Sets the snapshot expiration period for a virtual machine "data" in pool
"default" to seven days.
```

Options

```
  -p, --property   Set the key as a storage volume property
      --target     Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage volume* (page 897) - Manage storage volumes

`lxc storage volume show`

Show storage volume configurations

Synopsis

Description: Show storage volume configurations

```
lxc storage volume show [<remote>:]<pool> [<type>/]<volume>[/<snapshot>] [flags]
```

Examples

```
  Provide the type of the storage volume if it is not custom.
  Supported types are custom, image, container and virtual-machine.

  Add the name of the snapshot if type is one of custom, container or virtual-
machine.

  lxc storage volume show default data
      Will show the properties of a custom volume called "data" in the "default"
pool.

  lxc storage volume show default container/data
      Will show the properties of the filesystem for a container called "data" in
the "default" pool.

  lxc storage volume show default virtual-machine/data/snap0
      Will show the properties of snapshot "snap0" for a virtual machine called
"data" in the "default" pool.
```

Options

```
      --target   Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage volume* (page 897) - Manage storage volumes

`lxc storage volume snapshot`

Snapshot storage volumes

Synopsis

Description: Snapshot storage volumes

```
lxc storage volume snapshot [<remote>:]<pool> <volume> [<snapshot>] [flags]
```

Examples

```
  lxc storage volume snapshot default v1 snap0
         Create a snapshot of "v1" in pool "default" called "snap0".

  lxc storage volume snapshot default v1 snap0 < config.yaml
         Create a snapshot of "v1" in pool "default" called "snap0" with the
configuration from "config.yaml".
```

Options

```
      --no-expiry   Ignore any configured auto-expiry for the storage volume
      --reuse       If the snapshot name already exists, delete and create a new
one
      --target      Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage volume* (page 897) - Manage storage volumes

lxc storage volume unset

Unset storage volume configuration keys

Synopsis

Description: Unset storage volume configuration keys

```
lxc storage volume unset [<remote>:]<pool> [<type>/]<volume> <key> [flags]
```

Examples

```
  Provide the type of the storage volume if it is not custom.
  Supported types are custom, image, container and virtual-machine.

  lxc storage volume unset default data size
      Removes the size/quota of a custom volume "data" in pool "default".

  lxc storage volume unset default virtual-machine/data snapshots.expiry
      Removes the snapshot expiration period for a virtual machine "data" in pool
"default".
```

Options

```
  -p, --property   Unset the key as a storage volume property
      --target     Cluster member name
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc storage volume* (page 897) - Manage storage volumes

lxc version

Show local and remote versions

Synopsis

Description: Show local and remote versions

```
lxc version [<remote>:] [flags]
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD

`lxc warning`

Manage warnings

Synopsis

Description: Manage warnings

```
lxc warning [flags]
```

Options inherited from parent commands

```
      --debug           Show all debug messages
      --force-local     Force using the local unix socket
  -h, --help            Print help
      --project         Override the source project
  -q, --quiet           Don't show progress information
      --sub-commands    Use with help or --help to view sub-commands
  -v, --verbose         Show all information messages
      --version         Print version number
```

SEE ALSO

- *lxc* (page 690) - Command line client for LXD
- *lxc warning acknowledge* (page 915) - Acknowledge warning
- *lxc warning delete* (page 915) - Delete warning
- *lxc warning list* (page 916) - List warnings

- *lxc warning show* (page 917) - Show warning

`lxc warning acknowledge`

Acknowledge warning

Synopsis

Description: Acknowledge warning

```
lxc warning acknowledge [<remote>:]<warning-uuid> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc warning* (page 914) - Manage warnings

`lxc warning delete`

Delete warning

Synopsis

Description: Delete warning

```
lxc warning delete [<remote>:][<warning-uuid>] [flags]
```

Options

```
  -a, --all   Delete all warnings
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
```

(continues on next page)

(continued from previous page)

```
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc warning* (page 914) - Manage warnings

`lxc warning list`

List warnings

Synopsis

Description: List warnings

The -c option takes a (optionally comma-separated) list of arguments that control which warning attributes to output when displaying in table or csv format.

Default column layout is: utSscpLl

Column shorthand chars:

```
c - Count
l - Last seen
L - Location
f - First seen
p - Project
s - Severity
S - Status
u - UUID
t - Type
```

```
lxc warning list [<remote>:] [flags]
```

Options

```
-a, --all       List all warnings
-c, --columns   Columns (default "utSscpLl")
-f, --format    Format (csv|json|table|yaml|compact) (default "table")
```

Options inherited from parent commands

```
    --debug          Show all debug messages
    --force-local    Force using the local unix socket
-h, --help           Print help
    --project        Override the source project
-q, --quiet          Don't show progress information
    --sub-commands   Use with help or --help to view sub-commands
-v, --verbose        Show all information messages
    --version        Print version number
```

SEE ALSO

- *lxc warning* (page 914) - Manage warnings

`lxc warning show`

Show warning

Synopsis

Description: Show warning

```
lxc warning show [<remote>:]<warning-uuid> [flags]
```

Options inherited from parent commands

```
      --debug          Show all debug messages
      --force-local    Force using the local unix socket
  -h, --help           Print help
      --project        Override the source project
  -q, --quiet          Don't show progress information
      --sub-commands   Use with help or --help to view sub-commands
  -v, --verbose        Show all information messages
      --version        Print version number
```

SEE ALSO

- *lxc warning* (page 914) - Manage warnings

4.7. Implementation details

You don't need to be aware of the internal implementation details to use LXD. However, advanced users might be interested in knowing what happens internally.

4.7.1. Internals

Environment variables

The LXD client and daemon respect some environment variables to adapt to the user's environment and to turn some advanced features on and off.

> **Note**
>
> These environment variables are not available if you use the LXD snap.

Common

Name	Description
LXD_DIR	The LXD data directory
PATH	List of paths to look into when resolving binaries
http_proxy	Proxy server URL for HTTP
https_proxy	Proxy server URL for HTTPS
no_proxy	List of domains, IP addresses or CIDR ranges that don't require the use of a proxy

Client environment variable

Name	Description
EDITOR	What text editor to use
VISUAL	What text editor to use (if EDITOR isn't set)
LXD_CONF	Path to the LXC configuration directory
LXD_GLOBAL_CONF	Path to the global LXC configuration directory
LXC_REMOTE	Name of the remote to use (overrides configured default remote)

Server environment variable

Name	Description
LXD_EXEC_	Full path to the LXD binary (used when forking subcommands)
LXD_LXC_1	Path to the LXC template configuration directory
LXD_SECUF	If set to false, forces AppArmor off
LXD_UNPRI	If set to true, enforces that only unprivileged containers can be created. Note that any privileged containers that have been created before setting LXD_UNPRIVILEGED_ONLY will continue to be privileged. To use this option effectively it should be set when the LXD daemon is first set up.
LXD_OVMF_	Path to an OVMF build including OVMF_CODE.fd and OVMF_VARS.ms.fd (deprecated, please use LXD_QEMU_FW_PATH instead)
LXD_QEMU_	Path (or : separated list of paths) to firmware (OVMF, SeaBIOS) to be used by QEMU
LXD_IDMAF	Disable idmapped mounts support (useful when testing traditional UID shifting)
LXD_DEVMO	Path to be monitored by the device monitor. This is primarily for testing.
LXD_FSMON	Driver to be used for file system monitoring. This is primarily for testing.

UEFI variables for VMs

UEFI (Unified Extensible Firmware Interface) variables store and represent configuration settings of the UEFI firmware. See UEFI[277] for more information.

You can see a list of UEFI variables on your system by running `ls -l /sys/firmware/efi/efivars/`. Usually, you don't need to touch these variables, but in specific cases they can be useful to debug UEFI, SHIM, or boot loader issues in virtual machines.

[277] https://en.wikipedia.org/wiki/UEFI

To configure UEFI variables for a VM, use the *lxc config uefi* (page 756) command or the `/1.0/instances/<instance_name>/uefi-vars` endpoint.

For example, to set a variable to a value (hexadecimal):

CLI

API

```
lxc config uefi set <instance_name> <variable_name>-<GUID>=<value>
```

```
lxc query --request PUT /1.0/instances/<instance_name>/uefi-vars --data '{
  "variables": {
    "<variable_name>-<GUID>": {
      "attr": 3,
      "data": "<value>"
    },
  }
}'
```

See `PUT /1.0/instances/{name}/uefi-vars` for more information.

To display the variables that are set for a specific VM:

CLI

API

```
lxc config uefi show <instance_name>
```

```
lxc query --request GET /1.0/instances/<instance_name>/uefi-vars
```

See `GET /1.0/instances/{name}/uefi-vars` for more information.

Example

You can use UEFI variables to disable secure boot, for example.

> **Important**
>
> Use this method only for debugging purposes. LXD provides the *security.secureboot* (page 437) option to control the secure boot behavior.

The following command checks the secure boot state:

```
lxc config uefi get v1 SecureBootEnable-f0a30bc7-af08-4556-99c4-001009c93a44
```

A value of `01` indicates that secure boot is active. You can then turn it off with the following command:

```
lxc config uefi set v1 SecureBootEnable-f0a30bc7-af08-4556-99c4-001009c93a44=00
```

Daemon behavior

This specification covers some of the *LXD daemon* (page 345)'s behavior.

Startup

On every start, LXD checks that its directory structure exists. If it doesn't, it creates the required directories, generates a key pair and initializes the database.

Once the daemon is ready for work, LXD scans the instances table for any instance for which the stored power state differs from the current one. If an instance's power state was recorded as running and the instance isn't running, LXD starts it.

Signal handling

`SIGINT`, `SIGQUIT`, `SIGTERM`

For those signals, LXD assumes that it's being temporarily stopped and will be restarted at a later time to continue handling the instances.

The instances will keep running and LXD will close all connections and exit cleanly.

`SIGPWR`

Indicates to LXD that the host is going down.

LXD will attempt a clean shutdown of all the instances. After 30 seconds, it kills any remaining instance.

The instance `power_state` in the instances table is kept as it was so that LXD can restore the instances as they were after the host is done rebooting.

`SIGUSR1`

Write a memory profile dump to the file specified with `--memprofile`.

System call interception

LXD supports intercepting some specific system calls from unprivileged containers. If they're considered to be safe, it executes them with elevated privileges on the host.

Doing so comes with a performance impact for the syscall in question and will cause some work for LXD to evaluate the request and if allowed, process it with elevated privileges.

Enabling of specific system call interception options is done on a per-container basis through container configuration options.

Available system calls

`mknod` / `mknodat`

The `mknod` and `mknodat` system calls can be used to create a variety of special files.

Most commonly inside containers, they may be called to create block or character devices. Creating such devices isn't allowed in unprivileged containers as this is a very easy way to escalate privileges by allowing direct write access to resources like disks or memory.

But there are files which are safe to create. For those, intercepting this syscall may unblock some specific workloads and allow them to run inside an unprivileged containers.

The devices which are currently allowed are:

- OverlayFS whiteout (char 0:0)
- `/dev/console` (char 5:1)
- `/dev/full` (char 1:7)
- `/dev/null` (char 1:3)
- `/dev/random` (char 1:8)
- `/dev/tty` (char 5:0)
- `/dev/urandom` (char 1:9)
- `/dev/zero` (char 1:5)

All file types other than character devices are currently sent to the kernel as usual, so enabling this feature doesn't change their behavior at all.

This can be enabled by setting *`security.syscalls.intercept.mknod`* (page 439) to `true`.

`bpf`

The `bpf` system call is used to manage eBPF programs in the kernel. Those can be attached to a variety of kernel subsystems.

In general, loading of eBPF programs that are not trusted can be problematic as it can facilitate timing based attacks.

LXD's eBPF support is currently restricted to programs managing devices cgroup entries. To enable it, you need to set both *`security.syscalls.intercept.bpf`* (page 439) and *`security.syscalls.intercept.bpf.devices`* (page 439) to true.

`mount`

The `mount` system call allows for mounting both physical and virtual file systems. By default, unprivileged containers are restricted by the kernel to just a handful of virtual and network file systems.

To allow mounting physical file systems, system call interception can be used. LXD offers a variety of options to handle this.

`security.syscalls.intercept.mount` (page 440) is used to control the entire feature and needs to be turned on for any of the other options to work.

`security.syscalls.intercept.mount.allowed` (page 440) allows specifying a list of file systems which can be directly mounted in the container. This is the most dangerous option as it allows the user to feed data that is not trusted at the kernel. This can easily be used to crash the host system or to attack it. It should only ever be used in trusted environments.

`security.syscalls.intercept.mount.shift` (page 440) can be set on top of that so the resulting mount is shifted to the UID/GID map used by the container. This is needed to avoid everything showing up as `nobody`/`nogroup` inside of unprivileged containers.

The much safer alternative to those is *security.syscalls.intercept.mount.fuse* (page 440) which can be set to pairs of file-system name and FUSE handler. When this is set, an attempt at mounting one of the configured file systems will be transparently redirected to instead calling the FUSE equivalent of that file system.

As this is all running as the caller, it avoids the entire issue around the kernel attack surface and so is generally considered to be safe, though you should keep in mind that any kind of system call interception makes for an easy way to overload the host system.

sched_setscheduler

The `sched_setscheduler` system call is used to manage process priority.

Granting this may allow a user to significantly increase the priority of their processes, potentially taking a lot of system resources.

It also allows access to schedulers like `SCHED_FIFO` which are generally considered to be flawed and can significantly impact overall system stability. This is why under normal conditions, only the real root user (or global `CAP_SYS_NICE`) would allow its use.

setxattr

The `setxattr` system call is used to set extended attributes on files.

The attributes which are handled by this currently are:

- `trusted.overlay.opaque` (OverlayFS directory whiteout)

Note that because the mediation must happen on a number of character strings, there is no easy way at present to only intercept the few attributes we care about. As we only allow the attributes above, this may result in breakage for other attributes that would have been previously allowed by the kernel.

This can be enabled by setting *security.syscalls.intercept.setxattr* (page 441) to `true`.

sysinfo

The `sysinfo` system call is used by some distributions instead of `/proc/` entries to report on resource usage.

In order to provide resource usage information specific to the container, rather than the whole system, this syscall interception mode uses cgroup-based resource usage information to fill in the system call response.

Idmaps for user namespace

LXD runs safe containers. This is achieved mostly through the use of user namespaces which make it possible to run containers unprivileged, greatly limiting the attack surface.

User namespaces work by mapping a set of UIDs and GIDs on the host to a set of UIDs and GIDs in the container.

For example, we can define that the host UIDs and GIDs from 100000 to 165535 may be used by LXD and should be mapped to UID/GID 0 through 65535 in the container.

As a result a process running as UID 0 in the container will actually be running as UID 100000.

Allocations should always be of at least 65536 UIDs and GIDs to cover the POSIX range including root (0) and nobody (65534).

Kernel support

User namespaces require a kernel >= 3.12, LXD will start even on older kernels but will refuse to start containers.

Allowed ranges

> **If you installed LXD via snap**
>
> If you *installed LXD via the Snap Store* (page 28) (the recommended method), this section does not apply.
>
> The `lxd` daemon runs as root inside the snap environment and does not use the `newuidmap` or `newgidmap` utilities. Thus, the allowed ID ranges in `/etc/subuid` and `/etc/subgid` are ignored, and you don't need to set them.

On most hosts, LXD will check `/etc/subuid` and `/etc/subgid` for allocations for the `root` user and on first start, set the default profile to use the first 65536 UIDs and GIDs from that range.

If the range is shorter than 65536 (which includes no range at all), then LXD will fail to create or start any container until this is corrected.

If some but not all of `/etc/subuid`, `/etc/subgid`, `newuidmap` (path lookup) and `newgidmap` (path lookup) can be found on the system, LXD will fail the startup of any container until this is corrected as this shows a broken shadow setup.

If none of those files can be found, then LXD will assume a 1000000000 UID/GID range starting at a base UID/GID of 1000000.

This is the most common case and is usually the recommended setup when not running on a system which also hosts fully unprivileged containers (where the container runtime itself runs as a user).

Varying ranges between hosts

The source map is sent when moving containers between hosts so that they can be remapped on the receiving host.

Different idmaps per container

LXD supports using different idmaps per container, to further isolate containers from each other. This is controlled with two per-container configuration keys, *`security.idmap.isolated`* (page 436) and *`security.idmap.size`* (page 436).

Containers with `security.idmap.isolated` will have a unique ID range computed for them among the other containers with `security.idmap.isolated` set (if none is available, setting this key will simply fail).

Containers with `security.idmap.size` set will have their ID range set to this size. Isolated containers without this property set default to a ID range of size 65536; this allows for POSIX compliance and a `nobody` user inside the container.

To select a specific map, the `security.idmap.base` key will let you override the auto-detection mechanism and tell LXD what host UID/GID you want to use as the base for the container.

These properties require a container reboot to take effect.

Custom idmaps

LXD also supports customizing bits of the idmap, e.g. to allow users to bind mount parts of the host's file system into a container without the need for any UID-shifting file system. The per-container configuration key for this is *raw.idmap* (page 431), and looks like:

```
both 1000 1000
uid 50-60 500-510
gid 100000-110000 10000-20000
```

The first line configures both the UID and GID 1000 on the host to map to UID 1000 inside the container (this can be used for example to bind mount a user's home directory into a container).

The second and third lines map only the UID or GID ranges into the container, respectively. The second entry per line is the source ID, i.e. the ID on the host, and the third entry is the range inside the container. These ranges must be the same size.

This property requires a container reboot to take effect.

For non-snap installations of LXD, you might need to add an entry for the `root` user into `/etc/subuid` and/or `/etc/subgid` so the container is allowed to make use of it. See: *Allowed ranges* (page 923).

OVN implementation

Open Virtual Networks (OVN) is an open source Software Defined Network (SDN) solution. OVN is designed to be incredibly flexible. This flexibility comes at the cost of complexity. OVN is not prescriptive about how it should be used.

For LXD, the best way to think of OVN is as a toolkit. We need to translate networking concepts in LXD to their OVN analogue and instruct OVN directly, at a low level, what to do.

This document outlines LXD's approach to OVN in a basic setup. It does not yet cover load-balancers, peering, forwards, zones, or ACLs.

For more detailed documentation on OVN itself, please see:

- Overview of OVN and SDNs[278].
- OVN architectural overview[279].
- OVN northbound database schema documentation[280].
- OVN southbound database schema documentation[281].

[278] https://ubuntu.com/blog/data-centre-networking-what-is-ovn
[279] https://manpages.ubuntu.com/manpages/noble/man7/ovn-architecture.7.html
[280] https://manpages.ubuntu.com/manpages/noble/en/man5/ovn-nb.5.html
[281] https://manpages.ubuntu.com/manpages/noble/en/man5/ovn-sb.5.html

OVN concepts

This section outlines the OVN concepts that we use in LXD. These are usually represented in tables in the OVN northbound database.

Chassis

A chassis is where traffic physically ingresses into or egresses out of the virtual network. In LXD, there will usually be one chassis per cluster member. If LXD is configured to use OVN networking, then all members *can* be used as OVN chassis.

> **Note**
>
> If any cluster members have the role `ovn-chassis`, only those members are represented as chassis in the chassis group table (see below). If no members have the role, all cluster members are added to the chassis group.

Open vSwitch (OVS) Bridge

OVS bridges are used to connect virtual networks to physical ones and vice-versa. If the LXD daemon invokes OVS APIs, that means changes are being applied on the same host machine.

For each LXD cluster member there are two OVS bridges:

- The provider bridge. This is used when connecting the uplink network on the host to the external switch inside each OVN network.
- The integration bridge. This is used when connecting instances to the internal switch inside each OVN network.

Chassis group

A chassis group is an indirection between physical chassis and the virtual networks that use them. Each LXD OVN network has one chassis group. This allows us to, for example, set chassis priority on a per-network basis so that not all ingress/egress is happening on a single cluster member.

OVN underlay

The OVN underlay is the means by which networks are virtualized across cluster members. It is a Geneve tunnel which creates a layer 2 overlay network across layer 3 infrastructure. The OVN underlay is configured and managed by OVN.

Logical router

A logical router is a virtualized router. There is one per LXD OVN network. This handles layer 3 networking and additionally has associated NAT rules and security policies.

Logical switch

Logical routers cannot be directly connected to OVS bridges; for this, we use a logical switch. There are two logical switches per LXD OVN network:

- The external switch, which connects via logical switch port to a port on the logical router and to the provider OVS switch.
- The internal switch, which connects via logical switch port to a port on the logical router and to the integration bridge. This switch contains DHCP and IP allocation configuration.

Logical switch/router ports

When you create a logical router or switch in OVN, it doesn't initially have any ports. You need to create ports and then link them. For example, the internal logical switch and the logical router for a LXD OVN network are connected by:

1. Adding a logical router port to the logical router.
2. Adding a logical switch port to the internal logical switch.
3. Configuring the internal logical switch port as a router port and setting the logical router name.

Some configuration is applied directly at port level. For example, in a LXD OVN network, IPv6 router advertisement settings are applied on the logical router port for the internal switch. This is by design. It allows OVN to push configuration down to the port level so that packets are handled as quickly as possible.

Port groups

When a LXD OVN network is created, a port group will be created that is specific to that network. When instances are connected to the network, logical switch ports are created for them on the internal switch. These logical switch ports are added to the port group for the network. When a port group is created or updated in the OVN northbound database, the address set table is automatically populated. Address sets are used for managing access control lists (ACLs). By creating and maintaining the port group, we can easily select the whole network when managing ACLs.

OVN Uplink

An OVN network can specify an uplink network. That uplink network must be a managed network and be of type `physical` or `bridge`. From these managed network definitions LXD ascertains a `parent` interface to use for the uplink connectivity.

For managed `bridge` networks the interface is the name of the network itself.

For managed `physical` networks it is the per-cluster member value of the `parent` setting. The `parent` interface itself can have one of three types:

- Linux native bridge.
- OVS Bridge.
- Physical interface (or `bond` or `vlan`).

It is important to note that a `physical` managed network's `parent` interface can be any of these types, and that for a managed `bridge` network the parent interface can be either types of bridges.

Bridge (OVS)

A user can separately configure a managed bridge network with the `openvswitch bridge.driver`. An OVN network can be created with `network` set to the name of the managed bridge network. In this case LXD configures a bridge mapping on the OVS bridge to connect the OVN network:

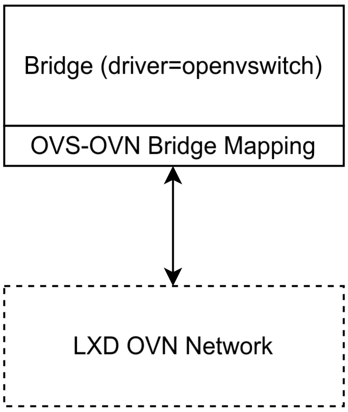


Fig. 3: OVN uplink OVS bridge

Physical

An OVN network can be created with `network` set to a physical network, where the physical network is essentially a database entry in LXD that tells it how to interact with an actual `parent` network device. In this case, an OVS bridge is created automatically. A bridge port connects the OVS bridge to the parent. A bridge mapping is used (as above) to connect the OVS bridge to the OVN network.

> **Note**
>
> When using a physical network as an uplink for OVN, any IP addresses on the parent interface will become defunct. The parent network must not have any assigned IP addresses.

Bridge (native)

A native Linux bridge can be used. In this case, we perform the same steps as in the physical network and additionally configure a `veth` pair. The `veth` pair is used so that the bridge can still be used for other purposes (since the bridge maintains its configuration). This is handy for development and testing but is not performant and should not be used in production.

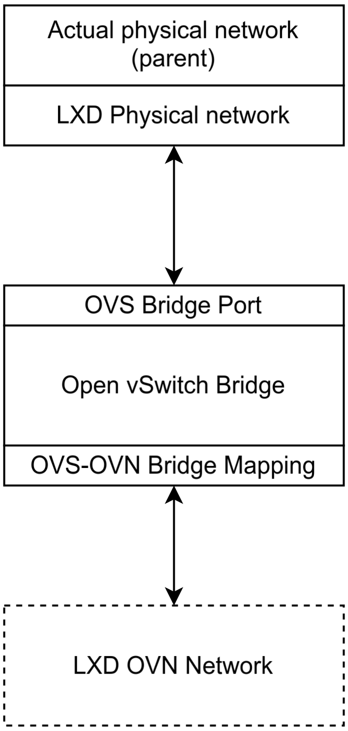


Fig. 4: OVN uplink physical

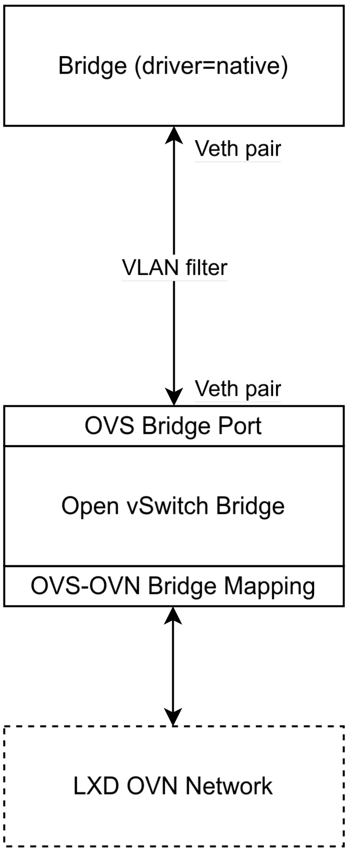


Fig. 5: OVN uplink native bridge

OVN Network

In the simplest case, a LXD OVN network has the below configuration:

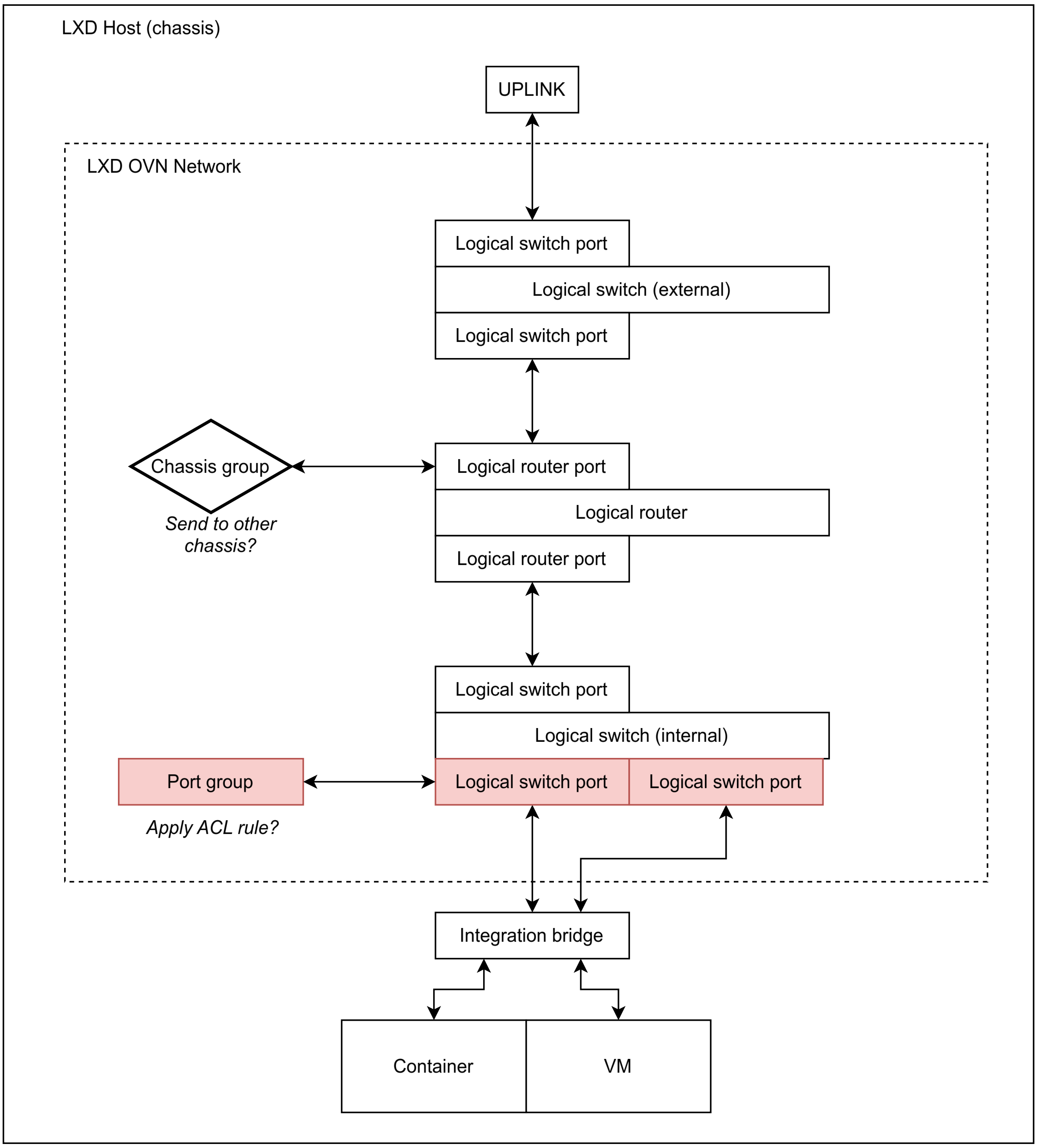


Fig. 6: LXD OVN network diagram

> **Note**
>
> This diagram does not show cross-cluster networks. This conceptual diagram should look the same on all cluster members. If the chassis group prioritizes another chassis for the uplink, the traffic is routed through that chassis.

Integration bridge

The cluster setting `network.ovn.integration_bridge` must contain the name of an OVS bridge that is used to connect instances to an OVN network via a NIC device. This OVS bridge must be pre-configured on all cluster members with the same name. Connectivity to the integration bridge differs between containers and virtual machines:

- Containers use a `veth` pair (similar to connecting to a native bridge uplink network).

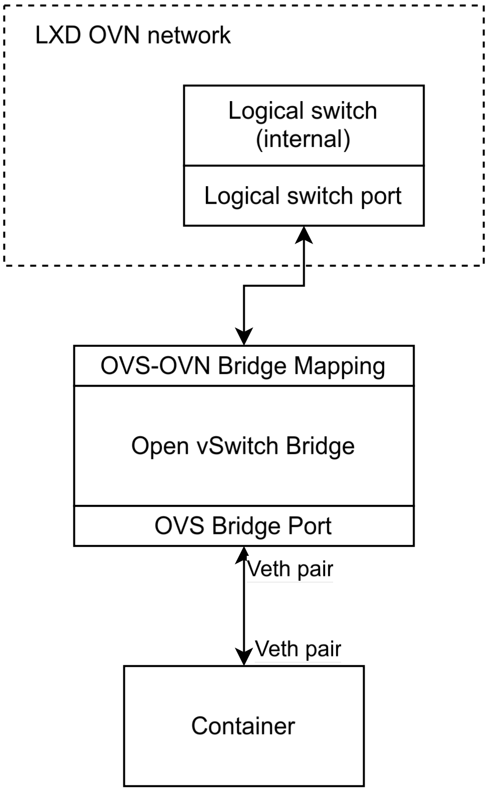


Fig. 7: Integration bridge connectivity with containers

- Virtual machines use a TAP device (this can be presented to QEMU as a device whereas a `veth` pair cannot).

VM live migration implementation

Live migration for virtual machines (page 136) in LXD is achieved by streaming instance state from a source QEMU (Quick Emulator) to a target QEMU. VM live migration is supported for all storage pool types.

API extension: `migration_vm_live`

Conceptual process

The live migration workflow varies depending on the type of storage pool used. The two key scenarios are non-shared storage and shared storage within a cluster (e.g., Ceph). If live state transfer is not supported by a target, a stateful stop is performed prior to migration.

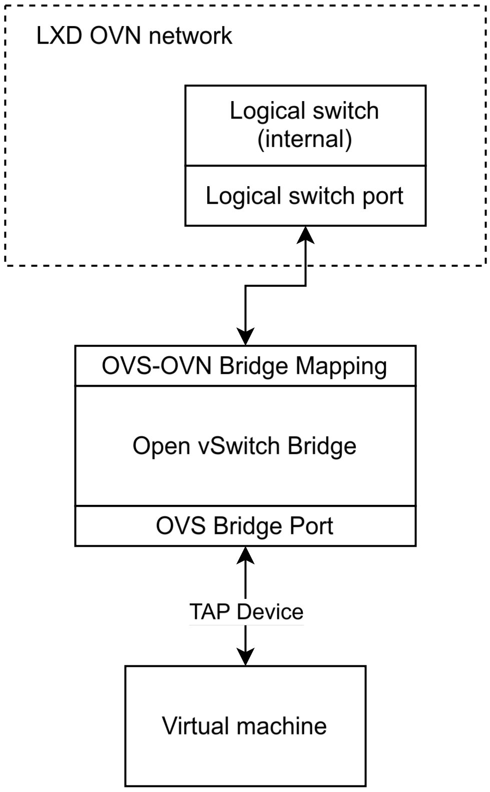


Fig. 8: Integration bridge connectivity with virtual machines

Live migration for non-shared storage

This process leverages the QEMU built-in Network Block Device (NBD) client-server mechanism to transfer the virtual machine's disk and memory state. Below is an overview of the steps:

1. Set up connection.
2. Determine migration type (shared or non-shared storage).
3. Set migration capabilities.
4. Non-shared storage preparation.
   1. Create and configure snapshot file for root disk writes during migration.
   2. Add the snapshot as a block device to the source VM.
   3. Redirect disk writes to the snapshot.
5. Storage transfer.
   1. For shared storage, we just perform checks at this point.
   2. For non-shared storage, set up an NBD listener and connect it to the target to transfer the disk.
6. Snapshot sync for non-shared storage to ensure consistency between source and target.
7. Transfer VM state to target.

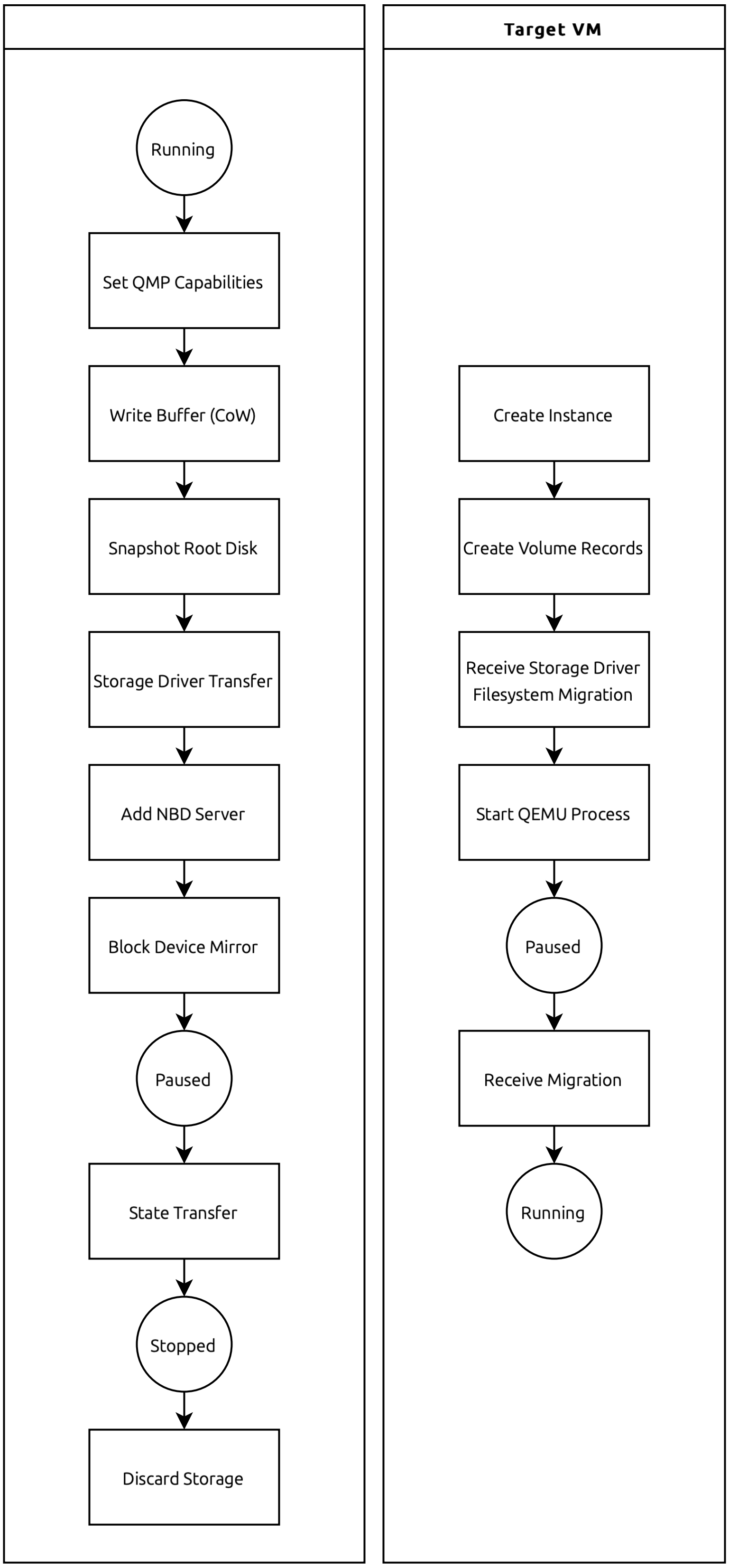


Fig. 9: Non-Shared Storage Migration

The state transitions during the process are shown below:

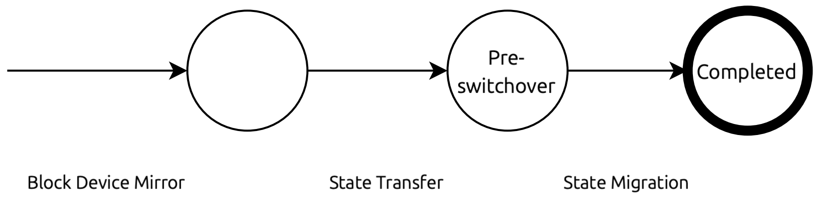


Fig. 10: Non-Shared Storage Migration State Transitions

Intra-cluster member live migration (Ceph shared storage pool)

For shared storage pools such as Ceph, disk data transfer is unnecessary. Instead, the process focuses on transferring the VM state through a dedicated migration socket:

1. Validate cluster state and storage pool readiness.
2. Notify the shared disks that they will be accessed from another system.
3. Pause the guest OS on the source VM and transfer the live state data over the migration socket.
4. Stop and delete source VM.
5. Start the target VM using the transferred state.

Migration API

Sending a `POST` request to `/1.0/instances/{name}` renames, moves an instance between pools, or migrates an instance to another server. In the push case, the returned operation metadata for migration is a background operation with progress data. For the pull case, it is a WebSocket operation with a number of secrets to be passed to the target server.

Live migration call stack

Below is a general overview of the key functions of the live migration call stack:

`lxd/lxd/instance_post.go`[282]

`instancePost`[283]

This function handles post requests to the `/1.0/instances` endpoint.

`lxd/lxd/migrate_instance.go`[284]

`Do`[285]

[282] https://github.com/canonical/lxd/blob/main/lxd/instance_post.go
[283] https://github.com/canonical/lxd/blob/main/lxd/instance_post.go#L74
[284] https://github.com/canonical/lxd/blob/main/lxd/migrate_instance.go
[285] https://github.com/canonical/lxd/blob/main/lxd/migrate_instance.go#L87

This function performs the migration operation on the source VM for the given state and operation. It sets up the necessary WebSocket connections for control, state, and filesystem, and then initiates the migration process.

`lxd/lxd/instance/drivers/driver_qemu.go`[286]

`MigrateSend`[287]

This function controls the sending of a migration, checking for stateful support, waiting for connections, performing checks, and sending a migration offer. When performing an intra-cluster same-name migration, steps are taken to prevent corruption of volatile device configuration keys during the start and stop of the instance on both source and target.

`migrateSendLive`[288]

This function performs the live migration send process:

1. Connect to the QEMU monitor: The function begins by establishing a connection to the QEMU monitor using `qmp.Connect`.
2. Define disk names: The function defines names for the root disk (`lxd_root`), the NBD target disk (`lxd_root_nbd`), and the snapshot disk (`lxd_root_snapshot`). These will be used later to manage the root disk and its snapshot during migration.
3. Check for shared storage: If the migration involves shared storage, the migration process can bypass the need for synchronizing the root disk. The function checks for this condition by verifying if `clusterMoveSourceName` is non-empty and the pool is remote.
4. Non-shared storage snapshot setup: If shared storage is not used, the function proceeds to set up a temporary snapshot of the root disk.
   1. Migration capabilities such as `auto-converge`, `pause-before-switchover`, and `zero-blocks` are set to optimize the migration process.
   2. The function creates a QCOW2 snapshot file of the root disk, which will store changes to the disk during migration.
   3. The snapshot file is opened for reading and writing, and the file descriptor is passed to QEMU.
   4. The snapshot is added as a block device to QEMU, ensuring that it is not visible to the guest OS.
   5. A snapshot of the root disk is taken using `monitor.BlockDevSnapshot`. This ensures that changes to the root disk are isolated during migration.
   6. Revert function: The revert function is used to clean up in case of failure. It ensures that the guest is resumed, and any changes made during snapshot creation are merged back into the root disk if migration fails
5. Shared storage setup: If shared storage is used, only the `auto-converge` migration capability is set, and no snapshot creation is necessary.
6. Perform storage transfer: The storage pool is migrated while the VM is still running. The `volSourceArgs.AllowInconsistent` flag is set to true to allow migration while the disk is in use. The migration checks are done by calling `pool.MigrateInstance`.

[286] https://github.com/canonical/lxd/blob/main/lxd/instance/drivers/driver_qemu.go
[287] https://github.com/canonical/lxd/blob/main/lxd/instance/drivers/driver_qemu.go#L6436
[288] https://github.com/canonical/lxd/blob/main/lxd/instance/drivers/driver_qemu.go#L6666

7. Notify shared disk pools: For each disk in the VM, the migration process checks if the disk belongs to a shared pool. If so, the disk is prepared for migration by calling `MigrateVolume` on the source disk.

8. Set up NBD listener and connection: If shared storage is not used, the function sets up a Unix socket listener for NBD connections. This listener handles the actual data transfer of the root disk from the source VM to the migration target.

9. Begin block device mirroring: After setting up the NBD connection, the function starts transferring the migration snapshot to the target disk by using `monitor.BlockDevMirror`.

10. Send stateful migration checkpoint: The function creates a pipe to transfer the state of the VM during the migration process. It writes the VMs state to the `stateConn` via the pipe, using `d.saveStateHandle` to handle the state transfer. Note that the source VMs guest OS is paused while the state is transferred. This ensures that the VMs state is consistent when the migration completes.

11. Finalize snapshot transfer: If non-shared storage is used, the function waits for the state transfer to reach the `pre-switchover` stage, ensuring that the guest remains paused during this process. Next, the function cancels the block job associated with the root snapshot to finalize the transfer and ensure that no changes are lost.

12. Completion: Once all transfers are complete, the function proceeds to finalize the migration process by resuming the target VM and ensuring that source VM resources are cleaned up. The source VM is stopped, and its storage is discarded.

Related topics

How-to guides:

- *Troubleshooting* (page 325)

Configuration options

instance

network

project

server

storage

sysctl