CONTENTS

1 In this documentation 3
2 Project and community 5
   Configuration options 623
LXD ([lksˈdɪː]) 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.
Tutorials

Start here: a hands-on introduction to LXD for new users

How-to guides

Step-by-step guides covering key operations and common tasks

Reference

Technical information - specifications, APIs, architecture

Explanation

Discussion and clarification of key topics, for example, Security
LXD is free software and developed under the Apache 2 license. 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.

- Code of Conduct
- Contribute to the project
- Release announcements
- Release tarballs
- Get support
- Watch tutorials and announcements on YouTube
- Discuss on IRC (see Getting started with IRC if needed)
- Ask and answer questions on the forum

2.1 Tutorials

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

2.1.1 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!
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*.

1. Install `snapd`:
   1. Run `snap version` to find out if snap is installed on your system:
      ```
      user@host:~$ snap version
      snap 2.59.4
      snapd 2.59.4
      series 16ubuntu 22.04
      kernel 5.15.0-73-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 in the Snapcraft documentation.

2. Enter the following command to install LXD:
   ```
   sudo snap install lxd
   ```
   If you get an error message that the 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. 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* for more information.

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:` image server.

You can list all images that are available on this server with:
```
```
See *Images* 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 *About containers and VMs* for information about the difference between the two instance types.

For managing instances, we use the LXD command line client `lxc`. See *About 1xd and 1xc* 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 22.04 image:
1. Launch a container called first using the Ubuntu 22.04 image:

```bash
lxc launch ubuntu:22.04 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:

```bash
lxc launch ubuntu:22.04 second
```

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

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

```bash
lxc copy first third
```

4. Launch a VM called ubuntu-vm using the Ubuntu 22.04 image:

```bash
lxc launch ubuntu:22.04 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:

```bash
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:

```bash
lxc start third
```

6. Query more information about each instance with:

```bash
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:

```bash
lxc stop second
```

2. Delete the second container:

```bash
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* and *How to manage instances* for more information.

**Configure instances**

There are several limits and configuration options that you can set for your instances. See *Instance options* 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:22.04 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:
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 of the root disk device for a VM:

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

```
user@host:~$ lxc exec ubuntu-vm -- df -h
```

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:

```
user@host:~$ lxc exec ubuntu-vm -- df -h
```

See [How to configure instances](#) and [Instance configuration](#) for more information.

### 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:
Canonical LXD

```bash
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* for more information.

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

1. Pull a file from the container:
   ```bash
   lxc file pull first/etc/hosts .
   ```

2. Add an entry to the file:
   ```bash
   echo "1.2.3.4 my-example" >> hosts
   ```

3. Push the file back to the container:
   ```bash
   lxc file push hosts first/etc/hosts
   ```

4. Use the same mechanism to access log files:
   ```bash
   lxc file pull first/var/log/syslog - | less
   ```

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

See *How to access files in an instance* for more information.

**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”:
   ```bash
   lxc snapshot first clean
   ```

2. Confirm that the snapshot has been created:
   ```bash
   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:
   ```bash
   lxc exec first -- rm /usr/bin/bash
   ```

4. Confirm the breakage:
   ```bash
   lxc exec first -- bash
   ```
You do not get a shell, because you deleted the bash command.

5. Restore the container to the state of the snapshot:
   ```bash
   lxc restore first clean
   ```

6. Confirm that everything is back to normal:
   ```bash
   lxc exec first -- bash
   exit
   ```

7. Delete the snapshot:
   ```bash
   lxc delete first/clean
   ```

See *Use snapshots for instance backup* for more information.

Next steps

Now that you’ve done your first experiments with LXD, check out the information in the *Getting started* section!

2.2 How-to guides

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

2.2.1 Clustering

The following how-to guides cover common operations related to clustering:

**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`, `parent`, `bgp.ipv4.nexthop` and `bgp.ipv6.nexthop`. See *Member configuration* 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:
   ```bash
   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`, you can see that the network is marked as “pending”.

2.2. How-to guides
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*.

### 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` (the address for internal cluster traffic) and `core.https_address` (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`, `size`, `zfs.pool_name`, `lvm.thinpool_name` and `lvm.vg_name`. See *Member configuration* 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`, you can see that the pool is marked as “pending”.

---

Chapter 2. Project and community
2. Run the following command to instantiate the storage pool on all cluster members:

```bash
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.*

**View member-specific pool configuration**

Running `lxc storage show <pool_name>` shows the cluster-wide configuration of the storage pool.

To view the member-specific configuration, use the `--target` flag. For example:

```bash
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>` 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`:

```bash
lxc storage volume create local vol1
```

To create a volume with the same name on another cluster member:

```bash
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:

```bash
lxc storage volume show local vol1 --target server1
lxc storage volume show local vol1 --target server2
```
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 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. Note, however, that this project is still in an early phase.

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.

- Setup password authentication on the cluster?
  Select no to use authentication tokens (recommended) or yes to use a trust password.

user@host:~$ 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]:Setup password authentication on the cluster? (yes/no) [default=no]: noDo you want to configure a new local storage pool? (yes/no) [default=no]: 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]:Would 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`.

**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`.
   - Do you have a join token?
     Select `yes` if you configured the bootstrap server to use **authentication tokens** (recommended) or `no` if you configured it to use a **trust password**.

2. Authenticate with the cluster.
   There are two alternative methods, depending on which authentication method you choose when configuring the bootstrap server.
   **Authentication tokens** (recommended)
   If you configured your cluster to use **authentication tokens**, you must generate a join token for each new member. To do so, run the following command on an existing cluster member (for example, the bootstrap server):

   ```bash
   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`). 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.

   **Trust password**
   If you configured your cluster to use a **trust password**, `lxd init` requires more information about the cluster before it can start the authorization process:
   - Specify a name for the new cluster member.
2. Provide the address of an existing cluster member (the bootstrap server or any other server you have already added).

3. Verify the fingerprint for the cluster.

4. If the fingerprint is correct, enter the trust password to authorize with the cluster.

3. Confirm that all local data for the server is lost when joining a cluster.

4. Configure server-specific settings (see Member configuration for more information).

You can accept the default values or specify custom values for each server.

Authentication tokens (recommended)

```
user@host:~$ sudo lxd init
Would you like to use LXD clustering? (yes/no) [default=no]: yes
What IP address or DNS name should be used to reach this server? [default=192.0.2.102]: 192.0.2.102
Are you joining an existing cluster? (yes/no) [default=no]: yes
Do you have a join token? (yes/no/[token]) [default=no]: yes
Please provide join token: eyJzZXJ2ZXJfbmFtZSI6InJwaTAxIiwiZmluZ2VycHJpbnQiOiIyZDIyNjJiZmExZDk6ZDZiMjk2Nzk0YjU0YzJ1YzdjOTMwNDAsZjIzNjM2NzE3NzYifQ==
Is this the correct fingerprint? (yes/no/[fingerprint]) [default=no]: yes
Cluster trust password:
```

All existing data is lost when joining a cluster, continue? (yes/no) [default=no]: yes
Choose "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]: yes
```

After the initialization process finishes, your server is added as a new cluster member. You can check this with lxc cluster list.

**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

The required contents of the preseed file depend on whether you want to use authentication tokens (recommended) or a trust password for authentication.

Authentication tokens (recommended)

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
- 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
```

Trust password

To enable clustering, the preseed file for the bootstrap server must contain the following fields:

```yaml
config:
  core.https_address: <IP_address_and_port>
  core.trust_password: <trust_password>
cluster:
  server_name: <server_name>
  enabled: true
```

Here is an example preseed file for the bootstrap server:
Join additional servers

The required contents of the preseed files depend on whether you configured the bootstrap server to use authentication tokens (recommended) or a trust password for authentication.

The preseed files for new cluster members require only a cluster section with data and configuration values that are specific to the joining server.

Authentication tokens (recommended)

The preseed file for additional servers must include the following fields:

```
cluster:
  enabled: true
  server_address: <IP_address_of_server>
  cluster_token: <join_token>
```

Here is an example preseed file for a new cluster member:

```
cluster:
  enabled: true
  server_address: 192.0.2.102:8443
  cluster_token: eyJzZXJ2ZXJfbmFtZSI6Im5vZGUyIiwiZmluZ2VycHJpbnQiOiIjZjlmNmVhMWIzYjhiNjgxNzQ1YTE1NTY2YjM3ZGUwOTUzNjRmMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMCIsImNhdWQiOiJjZjlmNmVhMWIzYjhiNjgxNzQ1YTE1NTY2YjM3ZGUwOTUzNjRmMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMCIiLCJpc3QiOiJjZjlmNmVhMWIzYjhiNjgxNzQ1YTE1NTY2YjM3ZGUwOTUzNjRmMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMCIsImlkIjoiMmNyb3NvY2tJcGhwaWQ6MzEwNzQ4ODQ0OTk4NzIyODQ2MTM5NzUwIiwiaWQiOiIjZjlmNmVhMWIzYjhiNjgxNzQ1YTE1NTY2YjM3ZGUwOTUzNjRmMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMCJ9
```

(continues on next page)
- 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"

Trust password

The preseed file for additional servers must include the following fields:

```yaml
cluster:
  server_name: <server_name>
  enabled: true
  cluster_address: <IP_address_of_bootstrap_server>
  server_address: <IP_address_of_server>
  cluster_password: <trust_password>
  cluster_certificate: <certificate>  # use this or cluster_certificate_path
  cluster_certificate_path: <path_to-certificate_file>  # use this or cluster_certificate
```

To create a YAML-compatible entry for the `cluster_certificate` key, run one the following commands on the bootstrap server:

- When using the snap:
  ```bash
  sed ':a;N;$!ba;s/\n/\n\n\n/g' /var/snap/lxd/common/lxd/cluster.crt
  ```
- Otherwise:
  ```bash
  sed ':a;N;$!ba;s/\n/\n\n\n/g' /var/lib/lxd/cluster.crt
  ```

Alternatively, copy the `cluster.crt` file from the bootstrap server to the server that you want to join and specify its path in the `cluster_certificate_path` key.

Here is an example preseed file for a new cluster member:

```yaml
cluster:
  server_name: server2
  enabled: true
  server_address: 192.0.2.102:8443
  cluster_address: 192.0.2.101:8443
  cluster_certificate: "-----BEGIN CERTIFICATE-----
opyQ1VRpAg2sV2C4W8irbNqeUsTeZZxhLqp4vN0XXBBrSqUCdPu1JXADV0kavg1l
2sXYoMobyV3K+RaIgsr10iHjacGiGCQT3YyNGGY/n5zgT/8xI0Dqvja0bNka6f6f
...

-----END CERTIFICATE-----"
  cluster_password: the_password
  member_config:
```

(continues on next page)
Use MicroCloud

Instead of setting up your LXD cluster manually, you can use MicroCloud 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
```

During the initialization process, MicroCloud detects the other servers, sets up OVN networking and prompts you to add disks to Ceph.

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 for more information.

How to manage a cluster

After your cluster is formed, use `lxc cluster list` to see a list of its members and their status:

```
user@host:~$ 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 |
server2 | https://192.0.2.102:8443 | database-standby | aarch64 | default | | ONLINE |
server3 | https://192.0.2.103:8443 | database-standby | aarch64 | default | | ONLINE |
```

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:
Configure your cluster

To configure your cluster, use `lxc config`. For example:

```
$ lxc config set cluster.max_voters 5
```

Keep in mind that some server configuration options are global and others are local. You can configure the global options on any cluster member, and the changes are propagated to the other cluster members through the distributed database. The local options are set only on the server where you configure them (or alternatively on the server that you target with `--target`).

In addition to the server configuration, there are a few cluster configurations that are specific to each cluster member. See Cluster member configuration for all available configurations.

To set these configuration options, use `lxc cluster set` or `lxc cluster edit`. For example:

```
$ lxc cluster set server1 scheduler.instance manual
```

Assign member roles

To add or remove a member role for a cluster member, use the `lxc cluster role` command. For example:

```
$ lxc cluster role add server1 event-hub
```

**Note:** You can add or remove only those roles that are not assigned automatically by LXD.

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 `lxc cluster edit` command.

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).

To do so, use the `lxc cluster evacuate` command. This command migrates all instances on the given server, moving them to other cluster members. The evacuated cluster member is then transitioned to an “evacuated” state, which prevents the creation of any instances on it.

You can control how each instance is moved through the `cluster.evacuate` instance configuration key. Instances are shut down cleanly, respecting the `boot.host_shutdown_timeout` configuration key.

When the evacuated server is available again, use the `lxc cluster restore` command to move the server back into a normal running state. This command also moves the evacuated instances back from the servers that were temporarily holding them.
Automatic evacuation

If you set the `cluster.healing_threshold` 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` 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, for example, a valid certificate obtained through ACME services (see [TLS server certificate](#) for more information). To do so, use the `lxc cluster update-certificate` command. This command replaces the certificate on all servers in your 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:22.04 c1 --target server2
```

You can launch instances on specific cluster members or on specific `cluster groups`.

If you do not specify a target, the instance is assigned to a cluster member automatically. See [Automatic placement of instances](#) 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.
Move an instance

You can move an existing instance to another cluster member. For example, to move 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 move existing LXD instances between servers* for more information.

To move 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 recover a cluster

It might happen that one or several members of your cluster go offline or become unreachable. In that case, no operations are possible on this member, and neither are operations that require a state change across all members. See *Offline members and fault tolerance* and *Automatic evacuation* for more information.

If you can bring the offline cluster members back or delete them from the cluster, operation resumes as normal. If this is not possible, there are a few ways to recover the cluster, depending on the scenario that caused the failure. See the following sections for details.

**Note:** When your cluster is in a state that needs recovery, most `lxc` commands do not work, because the LXD client cannot connect to the LXD daemon.

Therefore, the commands to recover the cluster are provided directly by the LXD daemon (`lxd`). Run `lxd cluster --help` for an overview of all available commands.

Recover from quorum loss

Every LXD cluster has a specific number of members (configured through `cluster.max_voters`) that serve as voting members of the distributed database. If you permanently 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. However, if at least one database member survives, it is possible to recover the cluster.

To do so, complete the following steps:

1. Log on to any surviving member of your cluster and run the following command:
   ```
sudo lxd cluster list-database
```
   This command shows which cluster members have one of the database roles.

2. Pick one of the listed database members that is still online as the new leader. Log on to the machine (if it differs from the one you are already logged on to).

3. Make sure that the LXD daemon is not running on the machine. For example, if you’re using the snap:
   ```
sudo snap stop lxd
```
4. Log on to all other cluster members that are still online and stop the LXD daemon.

5. On the server that you picked as the new leader, run the following command:

   ```
   sudo lxd cluster recover-from-quorum-loss
   ```

6. Start the LXD daemon again on all machines, starting with the new leader. 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.

### Recover cluster members with changed addresses

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, edit the cluster configuration on each member of the cluster and change the IP addresses or listening port numbers as required. You cannot remove any members during this process. The cluster configuration must contain the description of the full cluster, so you must do the changes for all cluster members on all cluster members.

You can edit the Member roles of the different 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.

Log on to each cluster member and complete the following steps:

1. Stop the LXD daemon. For example, if you’re using the snap:

   ```
   sudo snap stop lxd
   ```

2. Run the following command:

   ```
   sudo lxd cluster edit
   ```

3. Edit the YAML representation of the information that this cluster member has about the rest of the cluster:

   ```
   # Latest dqlite segment ID: 1234
   members:
   - id: 1
     name: server1
     address: 192.0.2.10:8443
     role: voter
   - id: 2
     name: server2
     address: 192.0.2.11:8443
     role: stand-by
   - id: 3
     # Internal ID of the member (Read-only)
   ```

(continues on next page)
You can edit the addresses and the roles.

After doing the changes on all cluster members, start the LXD daemon on all members again. For example, if you're using the snap:

```bash
sudo snap start lxd
```

The cluster should now be fully available again with all members reporting in. No information has been deleted from the database. All information about the cluster members and their instances is still there.

### 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` but still be part of the Raft configuration. To see the Raft configuration, run the following command:

```bash
lxd sql local "SELECT * FROM raft_nodes"
```

In that case, run the following command to remove the leftover node:

```bash
lxd cluster remove-raft-node <address>
```

### How to set up cluster groups

Cluster members can be assigned to *Cluster groups*. By default, all cluster members belong to the *default* group.

To create a cluster group, use the `lxc cluster group create` command. For example:

```bash
lxc cluster group create gpu
```

To assign a cluster member to one or more groups, use the `lxc cluster group assign` 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:

```bash
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:

```bash
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` 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` 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 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, but use the group name prefixed with `@` for the `--target` flag. For example:

```
lxc launch ubuntu:22.04 c1 --target=@gpu
```

Related topics

Explanation:

- About clustering

Reference:

- Cluster member configuration

2.2.2 Getting started

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

How to access the local LXD documentation

The latest version of the LXD documentation is available at documentation.ubuntu.com/lxd.

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. You can expose the server during initialization, or afterwards by setting the `core.https_address` 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, 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.
How to access the LXD web UI

Note: Starting with LXD 5.14, the LXD web UI is available as part of the LXD snap. See the LXD-UI GitHub repository for the source code.

The LXD web UI provides you with a graphical interface to manage your LXD server and instances. It is currently in an early stage and does not provide full functionality yet, but eventually, it will be an alternative to the LXD command-line client.

Complete the following steps to access the LXD web UI:

1. Enable the UI in the snap:
   ```
snap set lxd ui.enable=true
sudo systemctl reload snap.lxd.daemon
   ```

2. Make sure that your LXD server is exposed to the network. You can expose the server during initialization, or afterwards by setting the `core.https_address` server configuration option.

3. Access the UI in your browser by entering the server address (for example, `https://192.0.2.10:8443`).

   If you have not set up a secure **TLS server certificate**, 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.
4. 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. These steps include creating a set of certificates, adding the private key to your browser, and adding the public key to the server’s trust store.

See Remote API authentication for more information.
After setting up the certificates, you can start creating instances, editing profiles, or configuring your server.

### How to contribute to LXD

The LXD team appreciates contributions to the project, through pull requests, issues on the [GitHub repository](https://github.com), or discussions or questions on the [forum](https://community.canonical.com/lxd/).

Check the following guidelines before contributing to the project.
**Code of Conduct**

When contributing, you must adhere to the Code of Conduct, which is available at: https://github.com/canonical/lxd/blob/main/CODE_OF_CONDUCT.md

**License and copyright**

By default, any contribution to this project is made under the Apache 2.0 license. The author of a change remains the copyright holder of their code (no copyright assignment).

**Pull requests**

Changes to this project should be proposed as pull requests on GitHub at: https://github.com/canonical/lxd Proposed changes will then go through review there and once approved, be merged in the main branch.

**Commit structure**

Separate commits should be used for:

- API extension (api: Add XYZ extension, contains doc/api-extensions.md and shared/version.api.go)
- Documentation (doc: Update XYZ for files in doc/)
- API structure (shared/api: Add XYZ for changes to shared/api/)
- Go client package (client: Add XYZ for changes to client/)
- CLI (lxc/<command>: Change XYZ for changes to lxc/)
- Scripts (scripts: Update bash completion for XYZ for changes to scripts/)
- LXD daemon (lxd/<package>: Add support for XYZ for changes to lxd/)
- Tests (tests: Add test for XYZ for changes to tests/)

The same kind of pattern extends to the other tools in the LXD code tree and depending on complexity, things may be split into even smaller chunks.

When updating strings in the CLI tool (lxc/), you may need a commit to update the templates:

```bash
make i18n
git commit -a -s -m "i18n: Update translation templates" po/
```

When updating API (shared/api), you may need a commit to update the swagger YAML:

```bash
make update-api
git commit -s -m "doc/rest-api: Refresh swagger YAML" doc/rest-api.yaml
```

This structure makes it easier for contributions to be reviewed and also greatly simplifies the process of back-porting fixes to stable branches.
**Developer Certificate of Origin**

To improve tracking of contributions to this project we use the DCO 1.1 and use a “sign-off” procedure for all changes going into the branch.

The sign-off is a simple line at the end of the explanation for the commit which certifies that you wrote it or otherwise have the right to pass it on as an open-source contribution.

---

**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

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.

---

An example of a valid sign-off line is:

```
Signed-off-by: Random J Developer <random@developer.org>
```

Use a known identity and a valid e-mail address. Sorry, no anonymous contributions are allowed.

We also require each commit be individually signed-off by their author, even when part of a larger set. You may find `git commit -s` useful.
Contribute to the code

Follow the steps below to set up your development environment to get started working on new features for LXD.

Install LXD from source

To build the dependencies, follow the instructions in *Install LXD from source*.

Add your fork as a remote

After setting up your build environment, add your GitHub fork as a remote:

```
git remote add myfork git@github.com:<your_username>/lxd.git
git remote update
```

Then switch to it:

```
git checkout myfork/main
```

Build LXD

Finally, you should be able to run `make` inside the repository and build your fork of the project.

At this point, you most likely want to create a new branch for your changes on your fork:

```
git checkout -b [name_of_your_new_branch]
git push myfork [name_of_your_new_branch]
```

Important notes for new LXD contributors

- Persistent data is stored in the `LXD_DIR` directory, which is generated by `lxd init`. The `LXD_DIR` defaults to `/var/lib/lxd`, or `/var/snap/lxd/common/lxd` for snap users.
- As you develop, you may want to change the `LXD_DIR` for your fork of LXD so as to avoid version conflicts.
- Binaries compiled from your source will be generated in the `$(go env GOPATH)/bin` directory by default.
  - You will need to explicitly invoke these binaries (not the global `lxd` you may have installed) when testing your changes.
  - You may choose to create an alias in your `~/.bashrc` to call these binaries with the appropriate flags more conveniently.
- If you have a `systemd` service configured to run the LXD daemon from a previous installation of LXD, you may want to disable it to avoid version conflicts.
Contribute to the documentation

We want LXD to be as easy and straight-forward to use as possible. Therefore, we aim to provide documentation that contains the information that users need to work with LXD, that covers all common use cases, and that answers typical questions.

You can contribute to the documentation in various different ways. We appreciate your contributions!

Typical ways to contribute are:

- Add or update documentation for new features or feature improvements that you contribute to the code. We’ll review the documentation update and merge it together with your code.

- Add or update documentation that clarifies any doubts you had when working with the product. Such contributions can be done through a pull request or through a post in the Tutorials section on the forum. New tutorials will be considered for inclusion in the docs (through a link or by including the actual content).

- To request a fix to the documentation, open a documentation issue on GitHub. We’ll evaluate the issue and update the documentation accordingly.

- Post a question or a suggestion on the forum. We’ll monitor the posts and, if needed, update the documentation accordingly.

- Ask questions or provide suggestions in the #lxd channel on IRC. Given the dynamic nature of IRC, we cannot guarantee answers or reactions to IRC posts, but we monitor the channel and try to improve our documentation based on the received feedback.

Documentation framework

LXD’s documentation is built with Sphinx and hosted on Read the Docs.

It is written in Markdown with MyST extensions. For syntax help and guidelines, see the documentation cheat sheet (source).

For structuring, the documentation uses the Diátaxis 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/html/ 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:8001 to view the rendered documentation.

When you open a pull request, a preview of the documentation output is built automatically. To see the output, view the details for the docs/readthedocs.com:canonical-lxd check on the pull request.
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 as well with the following commands:

- Check the spelling: `make doc-spellcheck`
- Check the validity of links: `make doc-linkcheck`
- Check the Markdown formatting: `make doc-lint`
- Check for inclusive language: `make doc-woke`

Document configuration options

**Note:** We are currently in the process of moving the documentation of configuration options to code comments. At the moment, not all configuration options follow this approach.

The documentation of configuration options is extracted from comments in the Go code. Look for comments that start with `lxdmeta:generate` in the code.

When you add or change a configuration option, make sure to include the required documentation comment for it. See the `lxd-metadata README` file for information about the format.

Then run `make generate-config` to re-generate the `doc/config_options.txt` file. The updated file should be checked in.

The documentation includes sections from the `doc/config_options.txt` to display a group of configuration options. For example, to include the core server options:

```markdown
% Include content from [config_options.txt](config_options.txt)
```

```markdown
``` include} config_options.txt
```
```markdown
:start-after: <!-- config group server-core start -->
```
```markdown
:end-before: <!-- config group server-core end -->
```
```markdown
```

If you add a configuration option to an existing group, you don’t need to do any updates to the documentation files. The new option will automatically be picked up. You only need to add an include to a documentation file if you are defining a new group.

How to get support

LXD maintains different release branches in parallel.

**Long term support (LTS) releases**

The current LTS releases are LXD 5.0.x (snap channel 5.0/stable) and LXD 4.0.x (snap channel 4.0/stable).

The LTS releases follow the Ubuntu release schedule and are released every two years:

- LXD 5.0 is supported until June 2027 and gets frequent bugfix and security updates, but does not receive any feature additions. Updates to this release happen approximately every six months, but this schedule should be seen as a rough estimation that can change based on priorities and discovered bugs.
- LXD 4.0 is supported until June 2025.
• LXD 6.0 is planned for April 2024 and will be supported until June 2029.

Feature releases

The current feature release is LXD 5.x. It is available through the snap channels latest/stable, latest/candidate, and latest/edge, in addition to channels for the most recent specific releases (for example, 5.18/stable). See `sudo snap info lxd` for a full list of available channels.

Feature releases are pushed out about every month and contain new features as well as bugfixes. The normal support length for those releases is until the next release comes out. Some Linux distributions might offer longer support for particular feature releases that they decided to ship.

Support and community

The following channels are available for you to interact with the LXD community.

Bug reports

You can file bug reports and feature requests at: https://github.com/canonical/lxd/issues/new

Forum

A discussion forum is available at: https://discourse.ubuntu.com/c/lxd/

IRC

If you prefer live discussions, you can find us in #lxd on irc.libera.chat. See Getting started with IRC if needed.

Commercial support

Commercial support for LXD can be obtained through Canonical Ltd.

Documentation

The official documentation is available at: https://documentation.ubuntu.com/lxd/en/latest/

You can find additional resources on the website, on YouTube and in the Tutorials section in the forum.

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 About clustering and How to form a cluster)

A cluster combines several LXD servers. The cluster members share the same distributed database and can be managed uniformly using the LXD client (`lxc`) 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 and MAAS - Setting up LXD for VMs)

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 About networking and Network devices)

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 About storage pools, volumes and buckets and Storage drivers)

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 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 and Remote API authentication)

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.

You can choose to add client certificates to the server (manually or through tokens, the recommended way) or set a trust password.

**Automatic image update** (see About images)

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)

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, 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`.
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.

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, 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
  core.trust_password: sekret
  images.auto_update_interval: 6

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

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

# Profiles

(continues on next page)
profiles:
- name: default
devices:
  root:
  path: /
pool: data
type: disk
- name: test-profile
description: "Test profile"
cfg:
  limits.memory: 2GiB
devices:
  test0:
    name: test0
    nictype: bridged
    parent: lxd-my-bridge
type: nic

How to install LXD

The easiest way to install LXD is to install one of the available packages, but you can also install LXD from the sources. After installing LXD, make sure you have a lxd group on your system. Users in this group can interact with LXD. See Manage access to LXD for instructions.

Choose your release

LXD maintains different release branches in parallel.

Long term support (LTS) releases

The current LTS releases are LXD 5.0.x (snap channel 5.0/stable) and LXD 4.0.x (snap channel 4.0/stable).

The LTS releases follow the Ubuntu release schedule and are released every two years:

- LXD 5.0 is supported until June 2027 and gets frequent bugfix and security updates, but does not receive any feature additions. Updates to this release happen approximately every six months, but this schedule should be seen as a rough estimation that can change based on priorities and discovered bugs.
- LXD 4.0 is supported until June 2025.
- LXD 6.0 is planned for April 2024 and will be supported until June 2029.

Feature releases

The current feature release is LXD 5.x. It is available through the snap channels latest/stable, latest/candidate, and latest/edge, in addition to channels for the most recent specific releases (for example, 5.18/stable). See `sudo snap info lxd` for a full list of available channels.

Feature releases are pushed out about every month and contain new features as well as bugfixes. The normal support length for those releases is until the next release comes out. Some Linux distributions might offer longer support for particular feature releases that they decided to ship.

LTS releases are recommended for production environments, because they benefit from regular bugfix and security updates. However, there are no new features added to an LTS release, nor any kind of behavioral change.

To get all the latest features and monthly updates to LXD, use the feature release branch instead.
Install LXD from a package

The LXD daemon only works on Linux. The client tool (`lxc`) is available on most platforms.

**Linux**

The easiest way to install LXD on Linux is to install the *Snap package*, which is available for different Linux distributions.

If this option does not work for you, see the *Other installation options*.

**Snap package**

LXD publishes and tests *snap packages* that work for a number of Linux distributions (for example, Ubuntu, Arch Linux, Debian, Fedora, and OpenSUSE).

Complete the following steps to install the snap:

1. Check the LXD snap page on Snapcraft to see if a snap is available for your Linux distribution. If it is not, use one of the *Other installation options*.
2. Install `snapd`. See the *installation instructions* in the Snapcraft documentation.
3. Install the snap package. For the latest feature release, use:
   
   ```
   sudo snap install lxd
   ```

   For the LXD 5.0 LTS release, use:
   
   ```
   sudo snap install lxd --channel=5.0/stable
   ```

   For more information about LXD snap packages (regarding more versions, update management etc.), see *Managing the LXD snap*.

**Note:** On Ubuntu 18.04, if you previously had the LXD deb package installed, you can migrate all your existing data over with the following command:

```
sudo lxd.migrate
```

**Other installation options**

Some Linux distributions provide installation options other than the snap package.

**Alpine Linux**

To install the feature branch of LXD on Alpine Linux, run:

```apk
apk add lxd
```
Fedora

Fedora RPM packages for LXC/LXD are available in the COPR repository.

To install the LXD package for the feature branch, run:

```
 Pacman -S lxd
```

```
dnf copr enable ganto/lxc4
dnf install lxd
```

See the Installation Guide for more detailed installation instructions.

Gentoo

To install the feature branch of LXD on Gentoo, run:

```
emerge --ask lxd
```

Other operating systems

**Important:** The builds for other operating systems include only the client, not the server.

macOS

LXD publishes builds of the LXD client for macOS through Homebrew.

To install the feature branch of LXD, run:

```
brew install lxc
```

Windows

The LXD client on Windows is provided as a Chocolatey package. To install it:

1. Install Chocolatey by following the installation instructions.
2. Install the LXD client:

```
choco install lxc
```

You can also find native builds of the LXD client on GitHub:

- LXD client for Linux: `bin.linux.lxc.aarch64`, `bin.linux.lxc.x86_64`
- LXD client for Windows: `bin.windows.lxc.aarch64.exe`, `bin.windows.lxc.x86_64.exe`
- LXD client for macOS: `bin.macos.lxc.aarch64`, `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.
Install LXD from source

Follow these instructions if you want to build and install LXD from the source code.

We recommend having the latest versions of liblxc (>= 4.0.0 required) available for LXD development. Additionally, LXD requires Golang 1.18 or later to work. On Ubuntu, you can get those with:

```
sudo apt update
sudo apt install acl attr autoconf automake dnsmasq-base git golang libacl1-dev libcap-dev liblxc1 liblxc-dev libsqlite3-dev libtool libudev-dev libz4-dev libuv1-dev make pkg-config rsync squashfs-tools tar tcl xz-utils ebtables
```

**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 that, 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, and we are aware of it for Ubuntu 22.04/22.10. Ubuntu 23.04/23.10 does not have this problem.

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
```

To run the test suite, you’ll also need:

```
sudo apt install busybox-static curl gettext jq sqlite3 socat bind9-dnsutils
```

**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.

```

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 of libraft and libdqlite for LXD’s database setup.

```
tar zxfv 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.

```bash
user@host:~$ 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/ -I$(go env GOPATH)/deps/raft/include/"# export CGO_LDFLAGS="${CGO_LDFLAGS} -L$(go env GOPATH)/deps/dqlite/.libs/ -L$(go env GOPATH)/deps/raft/.libs/"# export LD_LIBRARY_PATH="$(go env GOPATH)/deps/dqlite/.libs/:$(go env GOPATH)/deps/raft/.libs/:${LD_LIBRARY_PATH}"# export CGO_LDFLAGS_ALLOW="(-Wl,-wrap,pthread_create)|(-Wl,-z,now)"
```

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:

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

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

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):

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

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.
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. See Access to the LXD daemon for more information.

If the lxd group is missing on your system, create it and restart the LXD daemon. You can then add trusted users to the group. Anyone added to this group will have full control over LXD.

Because group membership is normally only applied at login, you might need to either re-open your user session or use the newgrp lxd command in the shell you’re using to talk to LXD.

**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.

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 manage the LXD snap

Among other options, LXD is distributed as a snap. The benefit of packaging LXD as a snap is that it makes it possible to include all of LXD’s dependencies in one package, and that it allows LXD to be installed on many different Linux distributions. The snap ensures that LXD runs in a consistent environment.

Control updates of the snap

When running LXD in a production environment, you must make sure to have a suitable version of the snap installed on all machines of your LXD cluster.

Choose the right channel and track

Snaps come with different channels that define which release of a snap is installed and tracked for updates. See Channels and tracks in the snap documentation for detailed information.

Feature releases of LXD are available on the latest track. In addition, LXD provides tracks for the supported feature releases. See Choose your release for more information.

On all tracks, the stable risk level contains all fixes and features for the respective track, but it is only updated when the LXD team decides that a feature is ready and no issues have been revealed by users running the same revision on higher risk levels (edge and candidate).
When installing a snap, specify the channel as follows:

```
sudo snap install <snap_name> --channel=<channel>
```

For example:

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

To see all available channels of the LXD snap, run the following command:

```
snap info lxd
```

### Hold and schedule updates

By default, snaps are updated automatically. In the case of LXD, this can be problematic because all machines of a cluster must use the same version of the LXD snap.

Therefore, you should schedule your updates and make sure that all cluster members are in sync regarding the snap version that they use.

#### Schedule updates

There are two methods for scheduling when your snaps should be updated:

- You can hold snap updates for a specific time, either for specific snaps or for all snaps on your system. After the duration of the hold, or when you remove the hold, your snaps are automatically refreshed.
- You can specify a system-wide refresh window, so that snaps are automatically refreshed only within this time frame. Such a refresh window applies to all snaps.

#### Hold updates

You can hold snap updates for a specific time or forever, for all snaps or only for the LXD snap. If you want to fully control updates to your LXD deployment, you should put a hold on the LXD snap until you decide to update it.

Enter the following command to indefinitely hold all updates for the LXD snap:

```
sudo snap refresh --hold lxd
```

When you choose to update your installation, use the following commands to remove the hold, update the snap, and hold the updates again:

```
sudo snap refresh --unhold lxd
sudo snap refresh lxd --cohort="+
sudo snap refresh --hold lxd
```

See Hold refreshes in the snap documentation for detailed information about holding snap updates.

#### Specify a refresh window

Depending on your setup, you might want your snaps to update regularly, but only at specific times that don’t disturb normal operation.

You can achieve this by specifying a refresh timer. This option defines a refresh window for all snaps that are installed on the system.

For example, to configure your system to update snaps only between 8:00 am and 9:00 am on Mondays, set the following option:
You can use a similar mechanism (setting `refresh.hold`) to hold snap updates as well. However, in this case the snaps will be refreshed after 90 days, irrespective of the value of `refresh.hold`.

See Control updates with system options in the snap documentation for detailed information.

**Keep cluster members in sync**

The cluster members that are part of the LXD deployment must always run the same version of the LXD snap. This means that when the snap on one of the cluster members is refreshed, it must also be refreshed on all other cluster members before the LXD cluster is operational again.

Snap updates are delivered as progressive releases, which means that updated snap versions are made available to different machines at different times. This method can cause a problem for cluster updates if some cluster members are refreshed to a version that is not available to other cluster members yet.

To avoid this problem, use the `--cohort="+"` flag when refreshing your snaps:

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

This flag ensures that all machines in a cluster see the same snap revision and are therefore not affected by a progressive rollout.

**Use a Snap Store Proxy**

If you manage a large LXD cluster and you need absolute control over when updates are applied, consider installing a Snap Store Proxy.

The Snap Store Proxy is a separate application that sits between the snap client command on your machines and the snap store. You can configure the Snap Store Proxy to make only specific snap revisions available for installation.

See the Snap Store Proxy documentation for information about how to install and register the Snap Store Proxy.

After setting it up, configure the snap clients on all cluster members to use the proxy. See Configuring snap devices for instructions.

You can then configure the Snap Store Proxy to override the revision for the LXD snap:

```
sudo snap-proxy override lxd <channel>=<revision>
```

For example:

```
sudo snap-proxy 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 (see *Confine projects to specific LXD users* for more information).

See the LXD snap page for a list of available configuration options.

To set any of these options, use the following command:

```
sudo snap set lxd <key>=<value>
```

For example:

```
sudo snap set lxd daemon.user.group=lxd-users
```

To see all configuration options that are set on the snap, use the following command:

```
sudo snap get lxd
```

**Note:** This command returns only configuration options that have been explicitly set.

See *Configure snaps* in the snap documentation for more information about snap configuration options.

Start and stop the daemon

To start and stop the LXD daemon, you can use the `start` and `stop` commands of the snap:

```
sudo snap stop lxd
sudo snap start lxd
```

These commands are equivalent to running the corresponding `systemctl` commands:

```
sudo systemctl stop snap.lxd.daemon.service snap.lxd.daemon.unix.socket
sudo systemctl start snap.lxd.daemon.unix.socket; lxc list
```

Stopping the daemon also stops all running instances.

To restart the LXD daemon, use the following command:

```
sudo systemctl restart snap.lxd.daemon
```

Restarting the daemon stops all running instances. If you want to keep the instances running, reload the daemon instead:

```
sudo systemctl reload snap.lxd.daemon
```

**Note:** To restart the daemon, you can also use the snap commands. To stop all running instances and restart:

```
sudo snap restart lxd
```

To keep the instances running and reload:
However, there is currently a bug in snapd that causes undesired side effects when using the `snap restart` command. Therefore, we recommend using the `systemctl` commands instead.

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*

Explanation:
- *About containers and VMs*

Reference:
- *Requirements*

### 2.2.3 Images

The following how-to guides cover common operations related to images:

**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, use the `lxc image edit` command and edit the `profiles:` section:

```
profiles:
  - default
```

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.
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).

Copy an image from a remote

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` 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.

Import an image from files

If you have image files that use the required *Image format*, 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 file*)
- Building your own image using *distrobuilder* (see *Build an image*)
- Downloading image files from a *remote image server* (note that it is usually easier to *use the remote image* directly instead of downloading it to a file and importing it)

Import from the local file system

To import an image from the local file system, use the `lxc image import` command. This command supports both *unified images* (compressed file or directory) and *split images* (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` for all available flags.
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).

The image files must be provided as unified images (see Unified tarball).

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.

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.

To publish an image from an instance, make sure that the instance is stopped. Then 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` for a full list of available flags.

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.
Canonical 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` to edit)
- File templates (use `lxc config template` 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 `distrobuilder`.

See the `distrobuilder` documentation for instructions for installing and using the tool.

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 on another machine.

List available images

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 is used.

Filter available images

To filter the results that are displayed, specify a part of the alias or fingerprint after the command. For example, to show all Ubuntu 22.04 images, enter the following command:

```
lxc image list ubuntu: 22.04
```

You can specify several filters as well. For example, to show all Arm 64-bit Ubuntu 22.04 images, enter the following command:

```
lxc image list ubuntu: 22.04 arm64
```

To filter for properties other than alias or fingerprint, specify the filter in `<key>=<value>` format. For example:

```
lxc image list ubuntu: 22.04 architecture=x86_64
```
**View image information**

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:22.04`).

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 22.04 image, enter the following command:

```
lxc image get-property ubuntu:22.04 release
```

**Edit image properties**

To set a specific image property that is located under the `properties` key, enter the following command:

```
lxc image set-property <image_ID> <key>
```

**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>
```

**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).

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` or `publish` 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.

**Export an image to a file**

Images are located in the image store of your local server or a remote LXD server. You can export them to a file though. This method can be useful to back up image files or to transfer them to an air-gapped environment.

To export a container image to a file, enter the following command:

```
lxc image export [remote:]<image> [output_directory_path]
```

To export a virtual machine image to a file, add the `--vm` flag:

```
lxc image export [remote:]<image> [output_directory_path] --vm
```

See *Image format* for a description of the file structure used for the image.

**How to use remote images**

The `lxc` CLI command is pre-configured with several remote image servers. See *Remote image servers* for an overview.

**List configured remotes**

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

```
lxc remote list
```

Remote servers that use the `simple streams format` are pure image servers. Servers that use the `1xd` 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* 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* for instructions.
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> [flags]
```

Some authentication methods require specific flags (for example, use `lxc remote add <remote_name> <IP|FQDN|URL> --auth-type=candid` for Candid authentication). See Authenticate with the LXD server and Remote API authentication 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 password or token, depending on the authentication method used by the remote.

Reference an image

To reference an image, specify its remote and its alias or fingerprint, separated with a colon. For example:

```
ubuntu:22.04
images:ubuntu/22.04
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>
```
2.2.4 Instances

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

How to access files in an instance

You can manage files inside an instance using the LXD client 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, you can mount the instance’s file system onto the local machine.

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

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-container/etc/hosts
```

**Note:** The file must already exist on the instance. You cannot use the edit command to create a file on the instance.

Delete files from the instance

To delete a file from your instance, enter the following command:

```
lxc file delete <instance_name>/<path_to_file>
```
Pull files from the instance to the local machine

To pull a file from your instance to your local machine, enter the following command:

```bash
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:

```bash
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:

```bash
lxc file pull my-instance/var/log/syslog - | less
```

To pull a directory with all contents, enter the following command:

```bash
lxc file pull -r <instance_name>/<path_to_directory> <local_location>
```

Push files from the local machine to the instance

To push a file from your local machine to your instance, enter the following command:

```bash
lxc file push <local_file_path> <instance_name>/<path_to_file>
```

To push a directory with all contents, enter the following command:

```bash
lxc file push -r <local_location> <instance_name>/<path_to_directory>
```

Mount a file system from the instance

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):

```bash
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:

```bash
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`):
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.

## How to access the console

Use the `lxc console` command to attach to instance consoles. 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.

To get an interactive console, enter the following command:

```
lxc console <instance_name>
```

To show log output, 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
```
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.

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
```

How to add a routed NIC device to a virtual machine

When adding a routed NIC device 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:

```
lxc init ubuntu:22.04 jammy --vm
```

Then add the routed NIC device:

```
lxc config device add jammy eth0 nic nictype=routed parent=my-parent-network ipv4.address=192.0.2.2 ipv6.address=2001:db8::2
```

In this command, my-parent-network 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:

```
cat <<EOF | lxc config set jammy 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:
        - 192.0.2.2/32
        - 2001:db8::2/128

EOF
```

This netplan configuration adds the static link-local next-hop addresses (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.

**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.

You can then start your instance with:

```
lxc start jammy
```

**Note:** Before you start your instance, make sure that you have configured the parent network to enable proxy ARP/NDP.

### How to back up instances

There are different ways of backing up your instances:

- Use snapshots for instance backup
- Use export files for instance backup
- Copy an instance to a backup server

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* for instructions.

### 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*). 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

Use the following command to create a snapshot of an instance:

```
lxc snapshot <instance_name> [snapshot_name]
```

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.

View, edit or delete snapshots

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>
```

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` instance option.

For example, to configure daily snapshots, use the following command:

```
lxc config set <instance_name> snapshots.schedule @daily
```

To configure taking a snapshot every day at 6 am, use the following command:

```
lxc config set <instance_name> snapshots.schedule "0 6 * * *
```
When scheduling regular snapshots, consider setting an automatic expiry (`snapshots.expiry`) and a naming pattern for snapshots (`snapshots.pattern`). You should also configure whether you want to take snapshots of instances that are not running (`snapshots.schedule.stopped`).

**Restore an instance snapshot**

You can restore an instance to any of its snapshots. To do so, 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.

**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.

**Export an instance**

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 and makes them easily accessible.

- `--instance-only`

  By default, the export file contains all snapshots of the instance. Add this flag to export the instance without its snapshots.
Restore an instance from an export file

You can import an export file (for example, `/path/to/my-backup.tgz`) as a new instance. To do so, 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.

Copy an instance to a backup server

You can copy an instance to a secondary backup server to back it up.

See [How to move existing LXD instances between servers](#) for instructions.

How to configure instances

You can configure instances by setting Instance properties, Instance options, or by adding and configuring Devices.

See the following sections for instructions.

**Note:** To store and reuse different instance configurations, use profiles.

Configure instance options

You can specify instance options when you create an instance. Alternatively, you can update the instance options after the instance is created.

**CLI**

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

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

**API**

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.

**UI**

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*.

See *Instance options* 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**

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

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

**API**

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

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

**UI**

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* reference for information about which options are applied immediately while the instance is running.
Configure instance properties

CLI
To update instance properties after the instance is created, use the `lxc config set` 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
```

API
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>"}'}
```

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.

UI
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.

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 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
To add and configure an instance device for your instance, use the `lxc config device add` command.

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

lxc config device set <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. This is useful if you want to override device options for a device that is provided through a profile.

To remove a device, use the lxc config device remove command. See lxc config device --help for a full list of available commands.

API
To add and 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.

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

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"}}}'}

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

UI
The UI currently has limited support for devices.
To attach a device to your instance, you must first create it. Then you can update your instance configuration (in the same way as you configure instance options) to attach the device to the instance.

Note: Some of the devices that are displayed in the instance configuration are inherited from a profile or defined through a project. These devices cannot be edited 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.
**Display instance configuration**

**CLI**
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
```

**API**
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 /1.0/instances/<instance_name>
```

See [GET /1.0/instances/{name}](#) for more information.

**UI**
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**
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` command displays the full configuration including read-only instance properties. However, you cannot edit those properties. Any changes are ignored.

**API**
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.

**UI**
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 create instances

To create an instance, you can use either the lxc init or the lxc launch command. The lxc init command only creates the instance, while the lxc launch command creates and starts it.

Usage

Enter the following command to create a container:

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

Image

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 for more information.

Unless the image is available locally, you must specify the name of the image server and the name of the image (for example, ubuntu:22.04 for the official 22.04 Ubuntu image).

Instance name

Instance names must be unique within a LXD deployment (also within a cluster). See Instance properties for additional requirements.

Flags

See lxc launch --help or lxc init --help 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 for a device provided through a profile, or to specify an initial configuration for the root disk device
- **--profile** to specify a profile 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
Pass a configuration file

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:22.04 ubuntu-config < config.yaml
```

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

Examples

The following examples use `lxc launch`, but you can use `lxc init` in the same way.

Launch a container

To launch a container with an Ubuntu 22.04 image from the `images` server using the instance name `ubuntu-container`, enter the following command:

```
$ lxc launch ubuntu:22.04 ubuntu-container
```

Launch a virtual machine

To launch a virtual machine with an Ubuntu 22.04 image from the `images` server using the instance name `ubuntu-vm`, enter the following command:

```
$ lxc launch ubuntu:22.04 ubuntu-vm --vm
```

Or with a bigger disk:

```
$ lxc launch ubuntu:22.04 ubuntu-vm-big --vm --device root,size=30GiB
```

Launch a container with specific configuration options

To launch a container and limit its resources to one vCPU and 192 MiB of RAM, enter the following command:

```
$ lxc launch ubuntu:22.04 ubuntu-limited --config limits.cpu=1 --config limits.memory=192MiB
```
Launch a VM on a specific cluster member

To launch a virtual machine on the cluster member `server2`, enter the following command:

```
lxc launch ubuntu:22.04 ubuntu-container --vm --target server2
```

Launch 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 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 launch a container with this instance type, enter the following command:

```
lxc launch ubuntu:22.04 my-instance --type t2.micro
```

The list of supported clouds and instance types can be found at [https://github.com/dustinkirkland/instance-type](https://github.com/dustinkirkland/instance-type).

Launch a VM that boots from an ISO

To launch a VM that boots from an ISO, you must first create a VM. Let’s assume that we want to create a VM and install it from the ISO image. In this scenario, use the following command to create an empty VM:

```
lxc init iso-vm --empty --vm
```

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 <path-to-image.iso> iso-volume --type=iso
```

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

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

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

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

Once you're done in the serial console, you need to disconnect from the console using ctrl+a-q, and connect to the VGA console using the following command:
You should now see the installer. After the installation is done, you need to detach the custom ISO volume:

```
lxc storage volume detach default iso-volume iso-vm
```

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

**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**

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` to see all filter options.

**API**

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

```
lxc query /1.0/instances?recursion=2
```

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

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

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

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

See [GET /1.0/instances](#) for more information.

**UI**

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.

---

22. **How-to guides**
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
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
```

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

```
  lxc query /1.0/instances/<instance_name>
```

See [GET /1.0/instances/{name}](#) for more information.

UI
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
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](#) for more information.

API
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 /1.0/operations/<operation_ID>
```

Use the following query to monitor the state of the instance:
lxc query /1.0/instances/<instance_name>/state

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

UI
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.

**Stop an instance**

CLI
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.

API
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 /1.0/operations/<operation_ID>
```

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

```
lxc query /1.0/instances/<instance_name>/state
```

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

UI
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**

Enter the following command to delete an instance:

```
lxc delete <instance_name>
```

**API**

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.

**UI**

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` to true for the instance. See How to configure instances for instructions.

- In the CLI client, you can create an alias to be prompted for approval every time you use the `lxc delete` 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**

Enter the following command to rebuild the instance with a different image:
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`.

### API

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>", "server": "<server_URL>", "protocol": "simplestreams"}}
```

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.

### UI

Rebuilding an instance is not yet supported in the UI.

---

### How to run commands in an instance

LXD allows to run commands inside an instance using the LXD client, 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.

To run commands inside your instance, use the `lxc exec` command. By running a shell command (for example, `/bin/bash`), you can get shell access to your instance.

### Run commands inside your instance

To run a single command from the terminal of the host machine, use the `lxc exec` command:

```
$ lxc exec <instance_name> -- <command>
```

For example, enter the following command to update the package list on your container:

```
$ lxc exec ubuntu-container -- apt-get update
```
Execution mode

LXD can execute commands either interactively or non-interactively.

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.

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 through the following flags:

- `--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

Environment

You can pass environment variables to an exec session in the following two ways:

**Set environment variables as instance options**

To set the ENVAR environment variable to VALUE in the instance, set the `environment.ENVVAR` instance option (see `environment.*`):

```
 lxc config set <instance_name> environment.ENVVAR=VALUE
```

**Pass environment variables to the exec command**

To pass an environment variable to the exec command, use the `--env` flag. For example:

```
 lxc exec <instance_name> --env ENVVAR=VALUE -- <command>
```

In addition, LXD sets the following default values (unless they are passed in one of the ways described above):
Get shell access to your instance

If you want to run commands directly in your instance, run a shell command inside it. For example, enter the following command (assuming that the /bin/bash command exists in your instance):

```
lxc exec <instance_name> -- /bin/bash
```

By default, you are logged in as the root user. If you want to log in as a different user, enter the following command:

```
lxc exec <instance_name> -- su --login <user_name>
```

Note: Depending on the operating system that you run in your instance, you might need to create a user first.

To exit the instance shell, enter `exit` or press `Ctrl+d`.

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**
   Enter the following command to display the instance log:

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

   **Console log**
   Enter the following command to display the console log:

   ```
lxc console <instance_name> --show-log
```
Canonical LXD

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. Make sure to include the log files you collected.

Troubleshooting example

In this example, let's investigate a RHEL 7 system in which systemd cannot start.

```
user@host:~$ lxc console --show-log systemd
```

Console log: Failed to insert module 'autofs4'

```
Failed to insert module 'unix'
```

Failed 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** 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 configuration parameter. This is equivalent to setting init=/bin/bash on the Linux kernel command line.

```
lxc config set systemd raw.lxc 'lxc.init.cmd = /bin/bash'
```

Here is what it looks like:

```
user@host:~$ lxc config set systemd raw.lxc 'lxc.init.cmd = /bin/bash' user@host:~$ lxc start systemd
```

user@host:~$ lxc console --show-log systemd

Console log: [root@systemd /]

```
# ls
```

Now that the container has started, you can check it and see that things are not running as well as expected:

```
user@host:~$ lxc exec systemd bash [root@systemd ~]# ls [root@systemd ~]# mount
```

```
mount: failed to read mtab: No such file or directory
```

```
[root@systemd ~]# cd /proc
```

```
ls /proc/sys
```

```
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 use cloud-init

**cloud-init** 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 of the disk

See the Cloud-init documentation 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 have cloud-init support.
• Images from the images remote 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:ubuntu/22.04/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)
• cloud-init.user-data or user.user-data (see User data formats)
• cloud-init.network-config or user.network-config (see Network configuration)

For more information about the configuration options, see the cloud-init instance options, and the documentation for the LXD data source in the cloud-init documentation.

Vendor data and user data

Both vendor-data and user-data are used to provide cloud configuration data 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 for instructions.
How to configure cloud-init

To configure cloud-init for an instance, add the corresponding configuration options to a profile that the instance uses or directly to the instance configuration.

When configuring cloud-init directly for an instance, keep in mind that cloud-init runs only on the first start of the instance. That means that you must configure cloud-init before you start the instance. To do so, create the instance with lxc init instead of lxc launch, and then start it after completing the configuration.

YAML format for cloud-init configuration

The cloud-init options require YAML's literal style format. 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.

For example:

```
config:
  cloud-init.user-data: |
    #cloud-config
    package_upgrade: true
    packages:
      - package1
      - package2
```

Tip: See How to debug user data 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
```

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 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 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
```

Set the time zone

To set the time zone for the instance on instance creation, use the timezone key:

```yaml
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:

```yaml
config:
  cloud-init.user-data: |
  #cloud-config
  runcmd:
    - [touch, /run/cloud.init.ran]
```

Add a user account

To add a user account, use the `user` key. See the Including users and groups example in the cloud-init documentation for details about default users and which keys are supported.

```yaml
config:
  cloud-init.user-data: |
  #cloud-config
  user:
    - 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)

Example

To configure a specific network interface with a static IPv4 address and also use a custom name server, use the following configuration:

```yaml
config:
  cloud-init.network-config: |
    version: 1
    config:
      - type: physical
        name: eth1
        subnets:
          - type: static
            ipv4: true
```

(continues on next page)
How to use profiles

Profiles store a set of configuration options. They can contain instance options, devices 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

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>
```

Create an empty profile

Enter the following command to create an empty profile:

```
$ lxc profile create <profile_name>
```
Edit a profile

You can either set specific configuration options for a profile or edit the full profile in YAML format.

Set specific options for a profile

To set an instance option for a profile, use the `lxc profile set` 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` command. Specify the profile name, a device name, the device type and maybe device options (depending on the device type):

```
$ 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` command:

```
$ lxc profile device set <profile_name> <device_name> <device_option_key>=<device_option_value> <device_option_key>=<device_option_value> ...
```

Edit the full profile

Instead of setting each configuration option separately, you can provide all options at once in YAML format.

Check the contents of an existing profile or instance configuration for the required markup. 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
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`.

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
```

### Apply a profile to an instance

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>`

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> ...
```

### Remove a profile from an instance

Enter the following command to remove a profile from an instance:

```
$ lxc profile remove <instance_name> <profile_name>
```

### Related topics

Explanation:

- *About instances*

Reference:

- *Container runtime environment*
- *Instance configuration*

### 2.2.5 LXD server and client

The following how-to guides cover common operations related to the LXD server and the LXD client:
How to add command aliases

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` 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`. Run `lxc alias --help` to see all available subcommands.

How to add remote servers

Remote servers are a concept in the LXD command-line client. 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` 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` server configuration option must be set.

When adding the server, you must then authenticate with it using the chosen method for `Remote API authentication`. See `How to expose LXD to the network` for more information.

List configured remotes

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

```
lxc remote list
```

Remote servers that use the `simple streams format` are pure image servers. Servers that use the `1xd 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` 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> [flags]
```

Some authentication methods require specific flags (for example, use `lxc remote add <remote_name> <IP|FQDN|URL> --auth-type=candid` for Candid authentication). See [Authenticate with the LXD server](#) and [Remote API authentication](#) 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 password or token, depending on the authentication method used by the remote.

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` or `lxc remote set-url`), 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)
- `/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
```

(continues on next page)
How to configure the LXD server

See Server configuration 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 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

You can configure a server option with the following command:

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

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

```bash
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:

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

Display the server configuration

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

```bash
lxc config show
```

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

Edit the full server configuration

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

```bash
lxc config edit
```

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

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

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

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

```
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 for detailed information.

The recommended method is to add the client’s TLS certificate to the server’s trust store through a trust token. To authenticate a client using a trust token, complete the following steps:

1. On the server, enter the following command:

```
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.

2. 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 admin password, specify the generated 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.

See *Remote API authentication* for detailed information and other authentication methods.

**Related topics**

Explanation:

- *About lxd and lxc*
- *About the LXD database*

Reference:

- *Architectures*
- *Server configuration*
- *REST API*

### 2.2.6 Migration

LXD provides tools and functionality to migrate instances in different contexts.

**Migrate 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 machine. 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.

See *How to import physical or virtual machines to LXD instances* for more information.

**Migrate instances from LXC to LXD**

If you are using LXC and want to migrate all or some of your LXC containers to a LXD installation on the same machine, you can use the `lxc-to-lxd` tool. The tool analyzes the LXC configuration and copies the data and configuration of your existing LXC containers into new LXD containers.

See *How to migrate containers from LXC to LXD* for more information.

**Migrate existing LXD instances between servers**

The most basic kind of migration is if you have a LXD instance on one server and want to move it to a different LXD server. For virtual machines, you can do that as a live migration, which means that you can migrate your VM while it is running and there will be no downtime.

See *How to move existing LXD instances between servers* for more information.

The following how-to guides cover common operations related to migration:
How to import physical or virtual machines to LXD instances

LXD provides a tool (`lxd-migrate`) to create a LXD instance based on an existing disk or image.

You can run the tool on any Linux machine. It connects to a LXD server and creates a blank instance, which you can configure during or after the migration. The tool then copies the data from the disk or image that you provide to the 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* for more information. You can also override `Instance options`, the `storage pool` to be used and the size for the `storage volume`, and the `network` 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 image. 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.

**Tip:** If you want to convert a Windows VM from a foreign hypervisor (not from QEMU/KVM with Q35/virtio-scsi), you must install the `virtio-win` drivers to your Windows. Otherwise, your VM won’t boot.

1. Install `virt-v2v` version >= 2.3.4 (this is the minimal version that supports the `--block-driver` option).
2. Install the `virtio-win` package, or download the `virtio-win.iso` image and put it into the `/usr/share/virtio-win` folder.
3. You might also need to install `rhsrvany`.

Now you can use `virt-v2v` to convert images from a foreign hypervisor to raw images for LXD and include the required drivers:

```bash
# Example 1. Convert a vmdk disk image to a raw image suitable for lxd-migrate
sudo virt-v2v --block-driver virtio-scsi -o local -of raw -os ./os -i vmx ./test-vm.vmx

# Example 2. Convert a QEMU/KVM qcow2 image and integrate virtio-scsi driver
sudo virt-v2v --block-driver virtio-scsi -o local -of raw -os ./os -if qcow2 -i ../disk test-vm-disk.qcow2
```

You can find the resulting image in the `os` directory and use it with `lxd-migrate` on the next steps.

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` or `bin.linux.lxd-migrate.x86_64`) from the *Assets* section of the latest LXD release.
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`).

---

2.2. How-to guides
3. Make sure that the machine has `rsync` installed. If it is missing, install it (for example, with `sudo apt install rsync`).

4. Run the tool:

```bash
sudo ./bin.linux.lxd-migrate
```

The tool then asks you to provide the information required for the migration.

**Tip:** As an alternative to running the tool interactively, you can provide the configuration as parameters to the command. See `./bin.linux.lxd-migrate --help` for more information.

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. 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).

   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`. Then use the generated token to authenticate the tool.

4. Choose whether to create a container or a virtual machine. See About containers and VMs.

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, directly setting configuration options or changing storage or network settings.

   Alternatively, you can configure the new instance after the migration.

10. When you are done with the configuration, start the migration process.
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 size? [default=no]: yesPlease specify the storage size: 20GiB Instance to be created: Name: foo Project: default Type: container Source: / Storage pool: default Storage pool 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 pool 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

user@host:~$ 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: foo Please 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" Instance to be created: Name: foo Project: default Type: virtual-machine Source: ./virtual-machine.img Config: limits.cpu: "2" security.secureboot: "false" Instance to be created: Name: foo Project: default Type: virtual-machine Source: ./virtual-machine.img Config: limits.cpu: "2" security.secureboot: "false"
Change instance network Please pick one of the options above [default=1]: 1
Instance
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.

How to migrate containers from LXC to LXD

LXD provides a tool (lxc-to-lxd) that you can use to import LXC containers into your LXD server. The LXC containers must exist on the same machine as the LXD server.

The tool analyzes the LXC containers and migrates both their data and their configuration into new LXD containers.

Note: Alternatively, you can use the lxd-migrate tool within a LXC container to migrate it to LXD (see How to import physical or virtual machines to LXD instances). However, this tool does not migrate any of the LXC container configuration.

Get the tool

If you're using the snap, the lxc-to-lxd is automatically installed. It is available as lxd.lxc-to-lxd.

Otherwise, make sure that you have go (version 1.18 or later) installed and get the tool with the following command:

```
go install github.com/canonical/lxd/lxc-to-lxd@latest
```

Prepare your LXC containers

You can migrate one container at a time or all of your LXC containers at the same time.

Note: Migrated containers use the same name as the original containers. You cannot migrate containers with a name that already exists as an instance name in LXD.

Therefore, rename any LXC containers that might cause name conflicts before you start the migration process.

Before you start the migration process, stop the LXC containers that you want to migrate.

Start the migration process

Run sudo lxd.lxc-to-lxd [flags] to migrate the containers. (This command assumes that you are using the snap; otherwise, replace lxd.lxc-to-lxd with lxc-to-lxd, also in the following examples.)

For example, to migrate all containers:

```
sudo lxd.lxc-to-lxd --all
```

To migrate only the lxc1 container:

```
sudo lxd.lxc-to-lxd --containers lxc1
```

To migrate two containers (lxc1 and lxc2) and use the my-storage storage pool in LXD:
Canonical LXD

```bash
sudo lxd.lxc-to-lxd --containers lxc1,lxc2 --storage my-storage
```

To test the migration of all containers without actually running it:

```bash
sudo lxd.lxc-to-lxd --all --dry-run
```

To migrate all containers but limit the `rsync` bandwidth to 5000 KB/s:

```bash
sudo lxd.lxc-to-lxd --all --rsync-args --bwlimit=5000
```

Run `sudo lxd.lxc-to-lxd --help` to check all available flags.

**Note:** If you get an error that the `linux64` architecture isn’t supported, either update the tool to the latest version or change the architecture in the LXC container configuration from `linux64` to either `amd64` or `x86_64`.

### Check the configuration

The tool analyzes the LXC configuration and the configuration of the container (or containers) and migrates as much of the configuration as possible. You will see output similar to the following:

```
user@host:~$ sudo lxd.lxc-to-lxd --containers lxc1 Parsing LXC configurationChecking for unsupported LXC configuration keysChecking for existing containersChecking whether container has already been migratedValidating whether incomplete AppArmor support is enabledValidating whether mounting a minimal /dev is enabledValidating container rootfsProcessing network configurationProcessing storage configurationProcessing environment configurationProcessing container boot configurationProcessing container apparmor configurationProcessing container seccomp configurationProcessing container SELinux configurationProcessing container capabilities configurationProcessing container architecture configurationCreating containerTransferring container: lxc1: ...Container 'lxc1' successfully created
```

After the migration process is complete, you can check and, if necessary, update the configuration in LXD before you start the migrated LXD container.

### How to move existing LXD instances between servers

To move an instance from one LXD server to another, use the `lxc move` command:

```bash
lxc move [<source_remote>:]<source_instance_name> <target_remote>:[<target_instance_name>]
```

**Note:** When moving a container, you must stop it first. See [Live migration for containers](#) for more information.

When moving a virtual machine, you must either enable [Live migration for virtual machines](#) or stop it first.

Alternatively, you can use the `lxc copy` command if you want to duplicate the instance:

```bash
lxc copy [<source_remote>:]<source_instance_name> <target_remote>:[<target_instance_name>]
```

In both cases, 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. If you want to move the instance to a specific cluster member, specify it 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` for all available flags.

### Live migration

Live migration means migrating an instance 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 moved to another server while they are running, thus without any downtime.

To allow for live migration, you must enable support for stateful migration. To do so, ensure the following configuration:

- Set `migration.stateful` to `true` on the instance.
- Set `size.state` of the virtual machine’s root disk device to at least the size of the virtual machine's limits. memory setting.

#### Live migration for containers

For containers, there is limited support for live migration using CRIU (Checkpoint/Restore in Userspace). 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, move it over and 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:

```bash
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` 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`, or the maximum number of allowed iterations specified via `migration.incremental.memory.iterations` is reached, LXD instructs CRIU to perform a final memory dump and transfers it.
2.2.7 Networking

The following how-to guides cover common operations related to networking:

How to configure a network

To configure an existing network, use either the `lxc network set` and `lxc network unset` 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 for links to the configuration options for each network type.

There are separate commands to configure advanced networking features. See the following documentation:

- How to configure network ACLs
- How to configure network forwards
- How to configure network load balancers
- How to configure network zones
- How to create OVN peer routing relationships (OVN only)

How to configure LXD as a BGP server

**Note:** The BGP server feature is available for the Bridge network and the Physical network.

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` - the IP address for the BGP server
- `core.bgp_asn` - the ASN (Autonomous System Number) for the local server
- `core.bgp_routerid` - the unique identifier for the BGP server

For example, set the following values:

```bash
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 forwards

**Note:** Network forwards are available for the *OVN network* and the *Bridge network*.

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. There are two different ways how you can use network forwards in this case:

- 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.

Create a network forward

Use the following command to create a network forward:

```
$ lxc network forward create <network_name> <listen_address> [configuration_options...]
```

Each forward is assigned to a network. It requires a single external listen address (see *Requirements for listen addresses* for more information about which addresses can be forwarded, depending on the network that you are using).

You can specify an optional default target address by adding the `target_address=<IP_address>` configuration option. If you do, any traffic that does not match a port specification is forwarded to this address. Note that this target address must be within the same subnet as the network that the forward is associated to.

Forward properties

Network forwards have the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>listen_address</td>
<td>string</td>
<td>yes</td>
<td>IP address to listen on</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>no</td>
<td>Description of the network forward</td>
</tr>
<tr>
<td>config</td>
<td>string</td>
<td>no</td>
<td>Configuration options as key/value pairs (only target_address and user.* custom keys supported)</td>
</tr>
<tr>
<td>ports</td>
<td>port list</td>
<td>no</td>
<td>List of port specifications</td>
</tr>
</tbody>
</table>

2.2. How-to guides
Requirements for listen addresses

The requirements for valid listen addresses vary depending on which network type the forward is associated to.

**Bridge network**

- Any non-conflicting listen address is allowed.
- The listen address must not overlap with a subnet that is in use with another network.

**OVN network**

- Allowed listen addresses must be defined in the uplink network’s `ipv{n}.routes` settings or the project’s `restricted.networks.subnets` setting (if set).
- The listen address must not overlap with a subnet that is in use with another network.

Configure ports

You can add port specifications to the network forward to forward traffic from specific ports on the listen address to specific ports on the target address. This target address must be different from the default target address. It must be within the same subnet as the network that the forward is associated to.

Use the following command to add a port specification:

```
lxc network forward port add <network_name> <listen_address> <protocol> <listen_ports> [ <target_address> [ <target_ports> ] ]
```

You can specify a single listen port or a set of ports. 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.

Port properties

Network forward ports have the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>protocol</td>
<td>string</td>
<td>yes</td>
<td>Protocol for the port(s) (tcp or udp)</td>
</tr>
<tr>
<td>listen_port</td>
<td>string</td>
<td>yes</td>
<td>Listen port(s) (e.g. 80, 90-100)</td>
</tr>
<tr>
<td>target_address</td>
<td>string</td>
<td>yes</td>
<td>IP address to forward to</td>
</tr>
<tr>
<td>target_port</td>
<td>string</td>
<td>no</td>
<td>Target port(s) (e.g. 70, 80-90 or 90), same as listen_port if empty</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>no</td>
<td>Description of port(s)</td>
</tr>
</tbody>
</table>
Edit a network forward

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.

Delete a network forward

Use the following command to delete a network forward:

```
lxc network forward delete <network_name> <listen_address>
```

How to configure network ACLs

**Note:** Network ACLs are available for the *OVN NIC type*, the *OVN network* and the *Bridge network* (with some exceptions, see *Bridge limitations*).

Network ACLs (Access Control Lists) define traffic rules that allow controlling network access between different instances connected to the same network, and access 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 to all NICs connected to the network.

The instance NICs that have a particular ACL applied (either explicitly or implicitly through a network) make up a logical group, which can be referenced from other rules as a source or destination. See *ACL groups* for more information.

Create an ACL

Use the following command to create an ACL:

```
lxc network acl create <ACL_name> [configuration_options...]
```

This command creates an ACL without rules. As a next step, *add rules* to the ACL.

Valid network ACL names must adhere to the following rules:

- Names must be between 1 and 63 characters long.
- Names must be made up exclusively of letters, numbers and dashes from the ASCII table.
- Names must not start with a digit or a dash.
- Names must not end with a dash.
**ACL properties**

ACLs have the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>yes</td>
<td>Unique name of the network ACL in the project</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>no</td>
<td>Description of the network ACL</td>
</tr>
<tr>
<td>ingress</td>
<td>rule list</td>
<td>no</td>
<td>Ingress traffic rules</td>
</tr>
<tr>
<td>egress</td>
<td>rule list</td>
<td>no</td>
<td>Egress traffic rules</td>
</tr>
<tr>
<td>config</td>
<td>string set</td>
<td>no</td>
<td>Configuration options as key/value pairs (only user.* custom keys supported)</td>
</tr>
</tbody>
</table>

**Add or remove rules**

Each ACL contains two lists of rules:

- **Ingress** rules apply to inbound traffic going towards the NIC.
- **Egress** rules apply to outbound traffic leaving the NIC.

To add a rule to an ACL, use the following command, where `<direction>` can be either `ingress` or `egress`:

```
lxc network acl rule add <ACL_name> <direction> [properties...]```

This command adds a rule to the list for the specified direction.

You cannot edit a rule (except if you edit the full ACL), but you can delete rules with the following command:

```
lxc network acl rule remove <ACL_name> <direction> [properties...]```

You must either specify all properties needed to uniquely identify a rule or add `--force` to the command to delete all matching rules.

**Rule ordering and priorities**

Rules are provided as lists. However, the order of the rules in the list is not important and does not affect filtering.

LXD automatically orders the rules based on the `action` property as follows:

- drop
- reject
- allow
- Automatic default action for any unmatched traffic (defaults to reject, see *Configure default actions*).

This means that when you apply multiple ACLs to a NIC, there is no need to specify a combined rule ordering. If one of the rules in the ACLs matches, the action for that rule is taken and no other rules are considered.
Rule properties

ACL rules have the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>string</td>
<td>yes</td>
<td>Action to take for matching traffic (allow, reject or drop)</td>
</tr>
<tr>
<td>state</td>
<td>string</td>
<td>yes</td>
<td>State of the rule (enabled, disabled or logged), defaulting to enabled if not specified</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>no</td>
<td>Description of the rule</td>
</tr>
<tr>
<td>source</td>
<td>string</td>
<td>no</td>
<td>Comma-separated list of CIDR or IP ranges, source subject name selectors (for ingress rules), or empty for any</td>
</tr>
<tr>
<td>destination</td>
<td>string</td>
<td>no</td>
<td>Comma-separated list of CIDR or IP ranges, destination subject name selectors (for egress rules), or empty for any</td>
</tr>
<tr>
<td>protocol</td>
<td>string</td>
<td>no</td>
<td>Protocol to match (icmp4, icmp6, tcp, udp) or empty for any</td>
</tr>
<tr>
<td>source_port</td>
<td>string</td>
<td>no</td>
<td>If protocol is udp or tcp, then a comma-separated list of ports or port ranges (start-end inclusive), or empty for any</td>
</tr>
<tr>
<td>destination_</td>
<td>string</td>
<td>no</td>
<td>If protocol is udp or tcp, then a comma-separated list of ports or port ranges (start-end inclusive), or empty for any</td>
</tr>
<tr>
<td>icmp_type</td>
<td>string</td>
<td>no</td>
<td>If protocol is icmp4 or icmp6, then ICMP type number, or empty for any</td>
</tr>
<tr>
<td>icmp_code</td>
<td>string</td>
<td>no</td>
<td>If protocol is icmp4 or icmp6, then ICMP code number, or empty for any</td>
</tr>
</tbody>
</table>

Use selectors in rules

Note: This feature is supported only for the OVN NIC type and the OVN network.

The source field (for ingress rules) and the destination field (for egress rules) support using selectors instead of CIDR or IP ranges.

With this method, you can use ACL groups or network selectors to define rules for groups of instances without needing to maintain IP lists or create additional subnets.

ACL groups

Instance NICs that are assigned a particular ACL (either explicitly or implicitly through a network) make up a logical port group.

Such ACL groups are called subject name selectors, and they can be referenced with the name of the ACL in other ACL groups.

For example, if you have an ACL with the name foo, you can specify the group of instance NICs that are assigned this ACL as source with source=foo.
Network selectors

You can use network subject selectors to define rules based on the network that the traffic is coming from or going to. There are two special network subject selectors called @internal and @external. They represent network local and external traffic, respectively. For example:

```
source=@internal
```

If your network supports network peers, you can reference traffic to or from the peer connection by using a network subject selector in the format @<network_name>/<peer_name>. For example:

```
source=@ovn1/mypeer
```

When using a network subject selector, the network that has the ACL applied to it must have the specified peer connection. Otherwise, the ACL cannot be applied to it.

Log traffic

Generally, ACL rules are meant to control the network traffic between instances and networks. However, you can also use them to log specific network traffic, which can be useful for monitoring, or to test rules before actually enabling them.

To add a rule for logging, create it with the state=logged property. You can then display the log output for all logging rules in the ACL with the following command:

```
lx network acl show-log <ACL_name>
```

Edit an ACL

Use the following command to edit an ACL:

```
 lx network acl edit <ACL_name>
```

This command opens the ACL in YAML format for editing. You can edit both the ACL configuration and the rules.

Assign an ACL

After configuring an ACL, you must assign it to a network or an instance NIC.

To do so, add it to the security.acls list of the network or NIC configuration. For networks, use the following command:

```
 lx network set <network_name> security.acls="<ACL_name>"
```

For instance NICs, use the following command:

```
 lx config device set <instance_name> <device_name> security.acls="<ACL_name>"
```
Configure default actions

When one or more ACLs are applied to a NIC (either explicitly or implicitly through a 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 applied 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.

For example, to set the default action for inbound traffic to `allow` for all instances connected to a network, use the following command:

```
 lxc network set <network_name> security.acls.default.ingress.action=allow
```

To configure the same default action for an instance NIC, use the following command:

```
 lxc config device set <instance_name> <device_name> 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 are applied only on the boundary between the bridge and the LXD host. This means they can only be used to apply network policies for traffic going to or from external networks. They cannot be used for to create firewalls, thus firewalls that control traffic between instances connected to the same bridge.

- **ACL groups and network selectors** are not supported.

- When using the `iptables` firewall driver, you cannot use IP range subjects (for example, `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 once we have jumped into the ACL chain. Because of this, ACL rules cannot be used to block baseline service rules.

How to configure network zones

**Note:** Network zones are available for the **OVN network** and the **Bridge network**.

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` 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` 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`.

For example, running `dig @<DNS_server_IP> -p <DNS_server_PORT> axfr lxd.example.net` might give the following output:

```
user@host:~$ 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 192.0.2.192.0.2.20default-ovntest.uplink.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:

```
user@host:~$ 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` 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 1.0.0.0.1.0.0.8.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:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Required</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>peers.NAME.address</td>
<td>string</td>
<td>no</td>
<td>-</td>
<td>IP address of a DNS server</td>
</tr>
<tr>
<td>peers.NAME.key</td>
<td>string</td>
<td>no</td>
<td>-</td>
<td>TSIG key for the server</td>
</tr>
<tr>
<td>dns.nameservers</td>
<td>string</td>
<td>no</td>
<td>-</td>
<td>Comma-separated list of DNS server FQDNs (for NS records)</td>
</tr>
<tr>
<td>network.nat</td>
<td>bool</td>
<td>no</td>
<td>true</td>
<td>Whether to generate records for NAT-ed subnets</td>
</tr>
<tr>
<td>user.*</td>
<td>*</td>
<td>no</td>
<td>-</td>
<td>User-provided free-form key/value pairs</td>
</tr>
</tbody>
</table>

**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 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` 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:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>yes</td>
<td>Unique name of the record</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>no</td>
<td>Description of the record</td>
</tr>
<tr>
<td>entries</td>
<td>entry</td>
<td>no</td>
<td>A list of DNS entries</td>
</tr>
<tr>
<td>config</td>
<td>string</td>
<td>no</td>
<td>Configuration options as key/value pairs (only <code>user.*</code> custom keys supported)</td>
</tr>
</tbody>
</table>

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`), but you can delete entries with the following command:

```
lxc network zone record entry remove <network_zone> <record_name> <type> <value>
```
How to create a network

To create a managed network, use the `lxc network` 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:

<table>
<thead>
<tr>
<th>Network type</th>
<th>Documentation</th>
<th>Configuration options</th>
</tr>
</thead>
<tbody>
<tr>
<td>bridge</td>
<td>Bridge network</td>
<td>Configuration options</td>
</tr>
<tr>
<td>ovn</td>
<td>OVN network</td>
<td>Configuration options</td>
</tr>
<tr>
<td>macvlan</td>
<td>Macvlan network</td>
<td>Configuration options</td>
</tr>
<tr>
<td>sriov</td>
<td>SR-IOV network</td>
<td>Configuration options</td>
</tr>
<tr>
<td>physical</td>
<td>Physical network</td>
<td>Configuration options</td>
</tr>
</tbody>
</table>

Create a network

Use the following command to create a network:

```
$ lxc network create <name> --type=<network_type> [configuration_options...]
```

See *Network types* 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:

```
user@host:~$ lxc network create UPLINK --type=physical parent=br0 --target=vm01 Network UPLINK pending on member vm01 user@host:~$ lxc network create UPLINK --type=physical parent=br0 --target=vm02 Network UPLINK pending on member vm02 user@host:~$ lxc network create UPLINK --type=physical parent=br0 --target=vm03 Network UPLINK pending on member vm03 user@host:~$ lxc network create UPLINK --type=physical Network UPLINK created
```

Also see *How to configure networks for a cluster*.
Attach a network to an instance

After creating a managed network, you can attach it to an instance as a NIC device.

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
```

Attach the network as a device

The `lxc network attach` 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` for all available device options.

How to display IPAM information of a LXD deployment

IPAM (IPAddress 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/lxdb0 | 192.0.2.0/24 | network | true |                 |
| /1.0/networks/lxdb0 | 2001:db8::/32 | network | true |                 |
| /1.0/instances/u1 | 2001:db8::1/128 | instance | true | 00:16:3e:04:f0:95 |
```

(continues on next page)
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).

The following how-to guides apply to 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.
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:
Canonical LXD

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:

```bash
# 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="$(lxc network get lxdbr0 ipv4.address | sed 's|\.[0-9]/|/32|')"
CIDR6="$(lxc network get lxdbr0 ipv6.address | sed 's|:\[0-9]/|/128|')"
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 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 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 for more information.

To enable IPv4 forwarding before Docker starts, ensure that the following `sysctl` setting is enabled:

```bash
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:

```bash
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.

Use the following commands to explicitly allow egress network traffic flows from your LXD managed bridge interface:

```bash
iptables -I DOCKER-USER -i <network_bridge> -j ACCEPT
iptables -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:

```bash
iptables -I DOCKER-USER -i lxdbr0 -j ACCEPT
iptables -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` option (see *Configuration options*) must be set to `managed` or `dynamic` if you want to use this feature.

Depending on the configured `dns.domain`, you might need to disable DNSSEC in `resolved` to allow for DNS resolution. This can be done through the `DNSSEC` option in `resolved.conf`. 

---

2.2. How-to guides 115
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

You can automate the systemd-resolved DNS configuration, so that it is applied on system start and takes effect when LXD creates the network interface.

To do so, create a systemd unit file named `/etc/systemd/system/lxd-dns-<network_bridge>.service` with the following content:

```ini
[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`.

Then enable and start the service with the following commands:

```bash
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:

```bash
sudo systemctl status lxd-dns-<network_bridge>.service
```

You should see output similar to the following:

```bash
user@host:~$ 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>:
```

```bash
user@host:~$ 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 The following how-to guides apply to OVN networks only:
```

2.2. How-to guides
How to configure network load balancers

Note: Network load balancers are currently available for the OVN network.

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> [configuration_options..]
```

Each load balancer is assigned to a network. It requires a single external listen address (see Requirements for listen addresses for more information about which addresses can be load-balanced).

Load balancer properties

Network load balancers have the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>listen_address</td>
<td>string</td>
<td>yes</td>
<td>IP address to listen on</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>no</td>
<td>Description of the network load balancer</td>
</tr>
<tr>
<td>config</td>
<td>string set</td>
<td>no</td>
<td>Configuration options as key/value pairs (only user.* custom keys supported)</td>
</tr>
<tr>
<td>backends</td>
<td>backend list</td>
<td>no</td>
<td>List of backend specifications</td>
</tr>
<tr>
<td>ports</td>
<td>port list</td>
<td>no</td>
<td>List of port specifications</td>
</tr>
</tbody>
</table>
**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` setting (if set).
- 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 that the load balancer is associated to.

Use the following command to add a backend specification:

```
lxc network load-balancer backend add <network_name> <listen_address> <backend_name> <listen_ports> <target_address> [target_ports]
```

The target ports are optional. If not specified, the load balancer will use the listen ports for the backend for the backend target ports.

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:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>yes</td>
<td>Name of the backend</td>
</tr>
<tr>
<td>target_address</td>
<td>string</td>
<td>yes</td>
<td>IP address to forward to</td>
</tr>
<tr>
<td>target_port</td>
<td>string</td>
<td>no</td>
<td>Target port(s) (e.g. 70, 80-90 or 90), same as the <code>port</code>'s listen_port if empty</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>no</td>
<td>Description of backend</td>
</tr>
</tbody>
</table>

**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>...
```

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:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>protocol</td>
<td>string</td>
<td>yes</td>
<td>Protocol for the port(s) (tcp or udp)</td>
</tr>
<tr>
<td>listens_port</td>
<td>string</td>
<td>yes</td>
<td>Listen port(s) (e.g. 80, 90-100)</td>
</tr>
<tr>
<td>target_backend</td>
<td>backend list</td>
<td>yes</td>
<td>Backend name(s) to forward to</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>no</td>
<td>Description of port(s)</td>
</tr>
</tbody>
</table>

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 both the general configuration, backend and the port specifications.

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:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>yes</td>
<td>Name of the network peering on the local network</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>no</td>
<td>Description of the network peering</td>
</tr>
<tr>
<td>config</td>
<td>string set</td>
<td>no</td>
<td>Configuration options as key/value pairs (only user.* custom keys supported)</td>
</tr>
<tr>
<td>target_project</td>
<td>string</td>
<td>yes</td>
<td>Which project the target network exists in (required at create time)</td>
</tr>
<tr>
<td>target_network</td>
<td>string</td>
<td>yes</td>
<td>Which network to create a peering with (required at create time)</td>
</tr>
<tr>
<td>status</td>
<td>string</td>
<td>–</td>
<td>Status indicating if pending or created (mutual peering exists with the target network)</td>
</tr>
</tbody>
</table>

### 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 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
   
   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:22.04 c1
lxc config device override c1 eth0 network=ovntest
lxc start c1
   ```

5. Run `lxc list` to show the instance information:

   ```
user@host:$ lxc list
NAME | STATE | IPV4 | IPV6 | TYPE | SNAPSHOTS |
--- | --- | --- | --- | --- | --- |
c1 | RUNNING | 192.0.2.2 (eth0) | 2001:db8:fff3:5008:2b16:3eff:fed0:549f (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-nb-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 \
    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>` 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= \n    --<machine_name_1>
```

```
lxc network create UPLINK --type=physical parent=<uplink_interface> --target= \n    --<machine_name_2>
```

```
lxc network create UPLINK --type=physical parent=<uplink_interface> --target= \n    --<machine_name_3>
```

```
lxc network create UPLINK --type=physical parent=<uplink_interface> --target= \n    --<machine_name_4>
```

```
lxc network create UPLINK --type=physical \n    ipv4.ovn.ranges=<IP_range> \n    ipv6.ovn.ranges=<IP_range> \n    ipv4.gateway=<gateway> \n    ipv6.gateway=<gateway> \n    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 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:

```
lxc config set network.ovn.northbound_connection <ovn-northd-nb-db>
```

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:22.04 c1 --network my-ovn
lxc launch ubuntu:22.04 c2 --network my-ovn
lxc launch ubuntu:22.04 c3 --network my-ovn
lxc launch ubuntu:22.04 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` 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` configuration key, for example:
**Canonical LXD**

```bash
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`:

```bash
OVN_CTL_OPTS=" \
    --ovn-northd-log='-vsyslog:info --syslog-method=unix:/var/snap/lxd/common/lxd/syslog.\nsocket' \
    --ovn-nb-log='-vsyslog:info --syslog-method=unix:/var/snap/lxd/common/lxd/syslog.\nsocket' \
    --ovn-sb-log='-vsyslog:info --syslog-method=unix:/var/snap/lxd/common/lxd/syslog.\nsocket''
```

```bash
sudo systemctl restart ovn-central.service
```

**Related topics**

Explanation:

- *About networking*

Reference:

- *Networks*

### 2.2.8 Production setup

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

**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*. 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* 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* in the Snapcraft documentation for details.
**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).

See *How to back up instances* and *How to back up custom storage volumes* 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 move existing LXD instances between servers* and *How to move or copy storage volumes* 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. 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* and *Use export files for volume backup* 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* and *Use snapshots for volume backup* for more information.
Back up the database

While there is no trivial method to restore the contents of the LXD database, 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 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

If you're using the snap, the benchmarking tool is automatically installed. It is available as `lxd.benchmark`.

Otherwise, if you have installed LXD through your distribution’s package manager or built from source, the tool should be available as `lxd-benchmark`. If it isn’t, make sure that you have `go` (version 1.18 or later) installed and install 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. (This command assumes that you are using the snap; otherwise, replace `lxd.benchmark` with `lxd-benchmark`, also in the following examples.)

The benchmarking tool uses the current LXD configuration. 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.
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:22.04 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:22.04 local: --alias ubuntu
```

Remote image

If you want to include the download time in the overall result, specify a remote image (for example, `ubuntu:22.04`). 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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lxd.benchmark init --count 10 --privileged</code></td>
<td>Create ten privileged containers that use the latest Ubuntu image.</td>
</tr>
<tr>
<td><code>lxd.benchmark init --count 20 --parallel 4 images:alpine/edge</code></td>
<td>Create 20 containers that use the Alpine Edge image, using four parallel threads.</td>
</tr>
<tr>
<td><code>lxd.benchmark init 2d21da400963</code></td>
<td>Create one container that uses the local image with the fingerprint <code>2d21da400963</code>.</td>
</tr>
<tr>
<td><code>lxd.benchmark init --count 10 ubuntu</code></td>
<td>Create ten containers that use the image with the alias <code>ubuntu</code>.</td>
</tr>
</tbody>
</table>

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**

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.

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` for lists of available metrics.

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` command to query the `/1.0/metrics` endpoint:

```
user@host:~$ lxc query /1.0/metrics
```

```
# HELP lxd_cpu_seconds_total The total number of CPU time used in seconds.
# TYPE lxd_cpu_seconds_total counter
lxd_cpu_seconds_total{cpu="0",mode="system",name="u1",project="default",type="container"} 60.304517
lxd_cpu_seconds_total{cpu="0",mode="user",name="u1",project="default",type="container"} 145.647502
lxd_cpu_seconds_total{cpu="0",mode="iowait",name="vm",project="default",type="virtual-machine"} 4614.78
lxd_cpu_seconds_total{cpu="0",mode="irq",name="vm",project="default",type="virtual-machine"} 0
lxd_cpu_seconds_total{cpu="0",mode="idle",name="vm",project="default",type="virtual-machine"} 412762
lxd_cpu_seconds_total{cpu="0",mode="nice",name="vm",project="default",type="virtual-machine"} 35.
0
lxd_cpu_seconds_total{cpu="0",mode="softirq",name="vm",project="default",type="virtual-machine"} 2.41
lxd_cpu_seconds_total{cpu="0",mode="steal",name="vm",project="default",type="virtual-machine"} 9.84
lxd_cpu_seconds_total{cpu="0",mode="system",name="vm",project="default",type="virtual-machine"} 340.84
lxd_cpu_seconds_total{cpu="0",mode="user",name="vm",project="default",type="virtual-machine"} 261.25
```

```
# HELP lxd_cpu_effective_total The total number of effective CPUs.
# TYPE lxd_cpu_effective_total gauge
lxd_cpu_effective_total{name="u1",project="default",type="container"} 4
lxd_cpu_effective_total{name="vm",project="default",type="virtual-machine"} 0
```

```
# HELP lxd_disk_read_bytes_total The total number of bytes read.
# TYPE lxd_disk_read_bytes_total counter
lxd_disk_read_bytes_total{device="loop5",name="u1",project="default",type="container"} 2048
lxd_disk_read_bytes_total{device="loop3",name="vm",project="default",type="virtual-machine"} 353280...
```
Set up Prometheus

To gather and store the raw metrics, you should set up Prometheus. 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 server configuration option or the core.https_address 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
```

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:

```bash
# 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/
```

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):

```bash
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:

```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: 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:
Here is an example of a `prometheus.yml` configuration where multiple jobs are used to scrape the metrics of multiple LXD servers:

```
scrape_configs:
  - job_name: "lxd-hdc"
    metrics_path: '/1.0/metrics'
    params:
      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'

  - 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'
```

(continues on next page)
After editing the configuration, restart Prometheus (for example, `snap restart prometheus`) to start scraping.

**Set up a Grafana dashboard**

To visualize the metrics data, set up Grafana. LXD provides a Grafana dashboard that is configured to display the LXD metrics scraped by Prometheus.

Note: The dashboard requires Grafana 8.4 or later.

See the Grafana documentation for instructions on installing and signing in:

- Install Grafana
- Sign in to Grafana

Complete the following steps to import the LXD dashboard:

1. Configure Prometheus as the data source:
   1. Go to *Configuration > Data sources*.
   2. Click *Add data source*.
3. **Select Prometheus.**

4. In the *URL* field, enter `http://localhost:9090/`. 
5. Keep the default configuration for the other fields and click `Save & test`.

2. Import the LXD dashboard:
   1. Go to `Dashboards > Browse`.
   2. Click `New` and select `Import`.

3. In the `Import via grafana.com` field, enter the dashboard ID 19131.
4. Click **Load**.

5. In the **LXD** drop-down menu, select the Prometheus data source that you configured.

6. 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.
How to recover instances in case of disaster

LXD provides a tool for disaster recovery in case the LXD database 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:

```
user@host:~$ 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

user@host:~$ lxc network create lxdbr0
Network lxdbr0 created
user@host:~$ 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]: yes
Starting recovery...

user@host:~$ lxc list
+------+---------+------+------+-----------+-----------+
<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>IPV4</th>
<th>IPV6</th>
<th>TYPE</th>
<th>SNAPSHOTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>u1</td>
<td>STOPPED</td>
<td></td>
<td></td>
<td>CONTAINER</td>
<td>0</td>
</tr>
<tr>
<td>u2</td>
<td>STOPPED</td>
<td></td>
<td></td>
<td>CONTAINER</td>
<td>0</td>
</tr>
</tbody>
</table>

user@host:~$ lxc profile device add default eth0 nic network=lxdbr0 name=eth0
Device eth0 added to default
user@host:~$ lxc start u1
user@host:~$ lxc list

+------+---------+-------------------+---------------------------------------------+-----------+-----------+
<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>IPV4</th>
<th>IPV6</th>
<th>TYPE</th>
<th>SNAPSHOTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>u1</td>
<td>RUNNING</td>
<td>192.0.2.49 (eth0)</td>
<td>2001:db8:8b6:abfe:216:3eff:fe82:918e (eth0)</td>
<td>CONTAINER</td>
<td>0</td>
</tr>
<tr>
<td>u2</td>
<td>STOPPED</td>
<td></td>
<td></td>
<td>CONTAINER</td>
<td>0</td>
</tr>
</tbody>
</table>

Related topics

Explanation:

- About performance tuning

Reference:

- Provided metrics
- Server settings for a LXD production setup

### 2.2.9 Projects

The following how-to guides cover common operations related to projects:

#### How to confine projects to specific users

You can use projects to confine the activities of different users or clients. See *Confined projects in a multi-user environment* for more information.

How to confine a project to a specific user depends on the authentication method you choose.
Confine projects to specific TLS clients

You can confine access to specific projects by restricting the TLS client certificate that is used to connect to the LXD server. See `TLS client certificates` for detailed information.

To confine the access from the time the client certificate is added, you must either use token authentication or add the client certificate to the server directly. If you use password authentication, you can restrict the client certificate only after it has been added.

Use the following command to add a restricted client certificate:

```
Token authentication
lxc config trust add --projects <project_name> --restricted
```

Add client certificate

```
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>` or `lxc remote add <server_name> <server_address>`) and can only access the project or projects that have been specified.

To confine access for an existing certificate (either because the access restrictions change or because the certificate was added with a trust password), 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`.

**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.

Confine projects to specific RBAC roles

If you are using the Canonical RBAC service, the RBAC roles define what operations a user with that role can carry out. See `Role Based Access Control (RBAC)` for detailed information.

To use RBAC to confine a project, go to the respective project in the RBAC interface and assign RBAC roles to the different users or groups as required.

Confine projects to specific LXD users

If you use the LXD snap, 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` 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.

**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**

To create a project, use the `lxc project create` command.

You can specify configuration options by using the `--config` flag. See *Project configuration* 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 backups, enter the following command:

```
lxc project create my-restricted-project --config restricted=true --config restricted.backups=allow
```

**Tip:** When you create a project without specifying configuration options, `features.profiles` 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, use the `lxc profile device add` command to add a root disk device to the project's default profile.

**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**

To set a specific configuration option, use the `lxc project set` 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` 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.

**Edit the project**

To edit the full project configuration, use the `lxc project edit` command. For example:

```
lxc project edit my-project
```

**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, 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**

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:

```
user@host:$ lxc project list
+----------------------+--------+----------+-----------------+-----------------+----------+---------------+---------------------+---------+|
NAME | IMAGES | PROFILES | STORAGE 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` for more information.

**Switch projects**

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` command.

To switch to a different project, enter the following command:

```
lxc project switch <project_name>
```
Target a project

Instead of switching to a different project, you can target a specific project when running a command. Many LXD commands support the `--project` flag to run an action in a different project.

**Note:** You can target only projects that you have permission for.

The following sections give some typical examples where you would typically target a project instead of switching to it.

**List instances in a project**

To list the instances in a specific project, add the `--project` flag to the `lxc list` command. For example:

```
lxc list --project my-project
```

**Move an instance to another project**

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
```

**Copy a profile to another project**

If you create a project with the default settings, profiles are isolated in the project (`features.profiles` 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:

```
user@host:~$ lxc launch ubuntu:22.04 my-instance   Creating my-instanceError: 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, enter the following command:

```
lxc profile show default --project default | lxc profile edit default
```
Related topics

Explanation:
- About projects

Reference:
- Project configuration

2.2.10 Storage

The following how-to guides cover common operations related to storage:

How to back up custom storage volumes

There are different ways of backing up your custom storage volumes:
- Use snapshots for volume backup
- Use export files for volume backup
- Copy custom storage volumes

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. 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). 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

Use the following command to create a snapshot for a custom storage volume:

```
```
lxc storage volume snapshot <pool_name> <volume_name> [<snapshot_name>]
```

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.

View, edit or delete snapshots

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>
```

Schedule snapshots of a custom storage volume

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).

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 documentation for more information about those configuration options.
**Restore a snapshot of a custom storage volume**

You can restore a custom storage volume to the state of any of its snapshots. 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>
```

**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**

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 and makes them easily accessible.

- **--volume-only**
  By default, the export file contains all snapshots of the storage volume. Add this flag to export the volume without its snapshots.
Restore a custom storage volume from an export file

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.

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 for detailed information.

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>
```

Move instance storage volumes to another pool

To move an instance storage volume to another storage pool, make sure the instance is stopped. Then use the following command to move the instance to a different pool:

```
lxc move <instance_name> --storage <target_pool_name>
```

How to manage storage buckets and keys

See the following sections for instructions on how to create, configure, view and resize Storage buckets and how to manage storage bucket keys.

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` 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* for detailed information.

**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* 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* 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 quota size 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:
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 manage storage pools**

See the following sections for instructions on how to create, configure, view and resize *Storage pools.*

**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.

To create a storage pool, use the following command:

```
lxc storage create <pool_name> <driver> [configuration_options...]
```

Unless specified otherwise, LXD sets up loop-based storage with a sensible default size (20% of the free disk space, but at least 5 GiB and at most 30 GiB).

See the *Storage drivers* documentation for a list of available configuration options for each driver.
Examples

See the following examples for how to create a storage pool using different storage drivers.

Directory

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
```

Btrfs

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
```

LVM

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
```

ZFS

Create a loop-backed pool named pool11 (the ZFS zpool will also be called pool11):

```
lxc storage create pool11 zfs
```

Create a loop-backed pool named pool2 with the ZFS zpool name my-tank:
Use the existing ZFS zpool `my-tank` for pool3:

```bash
lxc storage create pool3 zfs source=my-tank
```

Use the existing ZFS dataset `my-tank/slice` for pool4:

```bash
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:

```bash
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`):

```bash
lxc storage create pool6 zfs source=/dev/sdX
```

Create a pool named `pool7` on `/dev/sdX` with the ZFS zpool name `my-tank`:

```bash
lxc storage create pool7 zfs source=/dev/sdX zfs.pool_name=my-tank
```

Ceph RBD

Create an OSD storage pool named `pool1` in the default Ceph cluster (named `ceph`):

```bash
lxc storage create pool1 ceph
```

Create an OSD storage pool named `pool2` in the Ceph cluster `my-cluster`:

```bash
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:

```bash
lxc storage create pool3 ceph ceph.osd.pool_name=my-osd
```

Use the existing OSD storage pool `my-already-existing-osd` for pool4:

```bash
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:

```bash
lxc storage create pool5 ceph source=rpl-pool ceph.osd.data_pool_name=ecpool
```

CephFS

**Note:** When using the CephFS driver, you must create a CephFS file system beforehand. This 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:

```bash
lxc storage create pool1 cephfs source=my-filesystem
```

Use the sub-directory `my-directory` from the `my-filesystem` file system for pool2:

```bash
```
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.

Therefore, you must first create 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.

For example, the following series of commands sets up a storage pool with the name `my-pool` at different locations and with different sizes on three cluster members:

```bash
user@host:~$ lxc storage create my-pool zfs source=/dev/sdX size=10GiB --target=vm01
Storage pool my-pool pending on member vm01
user@host:~$ lxc storage create my-pool zfs source=/dev/sdX size=15GiB --target=vm02
Storage pool my-pool pending on member vm02
user@host:~$ lxc storage create my-pool zfs source=/dev/sdY size=10GiB --target=vm03
Storage pool my-pool pending on member vm03
user@host:~$ lxc storage create my-pool zfs
Storage pool my-pool created
```

Also see [How to configure storage for a cluster](#).

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.

Configure storage pool settings

See the [Storage drivers](#) 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.

Use the following command to set configuration options for a storage pool:

```bash
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:
Canonical LXD

```
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>
```

**View storage pools**

You can display a list of all available storage pools and check their configuration.

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>
```

**Resize a storage pool**

If you need more storage, you can increase the size of your 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. You can only grow the pool (increase its size), not shrink it.

**How to manage storage volumes**

See the following sections for instructions on how to create, configure, view and resize Storage volumes.

**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 for detailed information.
Create the volume

Use the following command to create a custom storage volume of type block or filesystem in a storage pool:

```
lxc storage volume create <pool_name> <volume_name> [configuration_options...]
```

See the Storage drivers documentation for a list of available storage volume configuration options for each driver.

By default, custom storage volumes use the filesystem content type. To create a custom storage volume with the content type block, add the `--type` flag:

```
lxc storage volume create <pool_name> <volume_name> --type=block [configuration_options...]
```

To add a custom storage volume on a cluster member, add the `--target` flag:

```
lxc storage volume create <pool_name> <volume_name> --target=<cluster_member> [configuration_options...]
```

**Note:** 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 is different for Ceph-based storage pools (ceph and cephfs), where volumes are available from any cluster member.

To create a custom storage volume of type iso, use the import command instead of the create command:

```
lxc storage volume import <pool_name> <iso_path> <volume_name> --type=iso
```

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.

The following restrictions apply:

- Custom storage volumes of content type block or iso cannot be attached to containers, but only to virtual machines.
- To avoid data corruption, storage volumes of content type block should never be attached to more than one virtual machine at a time.
- Storage volumes of content type iso are always read-only, and can therefore be attached to more than one virtual machine at a time without corrupting data.
- File system storage volumes can’t be attached to virtual machines while they’re running.

For custom storage volumes with the content type filesystem, use the following command, where `<location>` is the path for accessing the storage volume inside the instance (for example, /data):

```
lxc storage volume attach <pool_name> <filesystem_volume_name> <instance_name> <location>
```

Custom storage volumes with the content type block do not take a location:

```
lxc storage volume attach <pool_name> <block_volume_name> <instance_name>
```

By default, the custom storage volume is added to the instance with the volume name as the device name. If you want to use a different device name, you can add it to the command:
Attach the volume as a device

The `lxc storage volume attach` command is a shortcut for adding a disk device to an instance. Alternatively, you can add a disk device for the storage volume in the usual way:

```
lxc config device add <instance_name> <device_name> disk pool=<pool_name> source=<volume_name> [path=<location>]
```

When using this way, you can add further configuration to the command if needed. See `disk device` for all available device options.

Configure I/O limits

When you attach a storage volume to an instance as a disk device, you can configure I/O limits for it. To do so, set the `limits.read`, `limits.write` or `limits.max` properties to the corresponding limits. See the `Type: disk` 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.

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 or images.

To do so, you must set the corresponding server configuration:

- To use a custom volume to store the backup tarballs:
  ```
lxc config set storage.backups_volume <pool_name>/<volume_name>
  ```

- To use a custom volume to store the image tarballs:
  ```
lxc config set storage.images_volume <pool_name>/<volume_name>
  ```
Configure storage volume settings

See the Storage drivers documentation for the available configuration options for each storage driver.

Use the following command to set configuration options for a storage volume:

```
lxc storage volume set <pool_name> [/<volume_type>/]<volume_name> <key> <value>
```

The default storage volume type is custom, so you can leave out the <volume_type>/ when configuring a custom storage volume.

For example, to set the size of your custom storage volume `my-volume` to 1 GiB, use the following command:

```
lxc storage volume set my-pool my-volume size=1GiB
```

To set the snapshot expiry time for your 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 by using the following command:

```
lxc storage volume edit <pool_name> [/<volume_type>/]<volume_name>
```

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, thus volume.<VOLUME_CONFIGURATION>=<VALUE>.

This value is then used for all new storage volumes in the pool, unless it is set explicitly for a volume or an instance. In general, the defaults set on a storage pool level (before the volume was created) can be overridden through the volume configuration, and the volume configuration can be overridden through the instance configuration (for storage volumes of type container or virtual-machine).

For example, to set a default volume size for a storage pool, use the following command:

```
lxc storage set [<remote>://]<pool_name> volume.size <value>
```

View storage volumes

You can display a list of all available storage volumes in a storage pool and check their configuration.

To list all available storage volumes in a storage pool, use the following command:

```
lxc storage volume list <pool_name>
```

To display the storage volumes for all projects (not only the default project), add the --all-projects flag.

The resulting table contains the storage volume type and the content type for each storage volume in the pool.

**Note:** Custom storage volumes might 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 <pool_name> [<volume_type>/]<volume_name>
```

To show state information about a specific volume, use the following command:

```
lxc storage volume info <pool_name> [<volume_type>/]<volume_name>
```

In both commands, the default storage volume type is custom, so you can leave out the `<volume_type>/` when displaying information about a custom storage volume.

**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 resize a storage volume, set its size configuration:

```
lxc storage volume set <pool_name> <volume_name> size <new_size>
```

**Important:**

- Growing a storage volume usually works (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.

**How to move or copy storage volumes**

You can *copy* or *move* 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* 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**

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.

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.
Move or rename custom storage volumes

Before you can move or rename a custom storage volume, all instances that use it must be stopped.

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.

Copy or move between cluster members

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 move 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.

Copy or move between projects

Add the `--target-project` to copy or move a custom storage volume to a different project.

Copy or move between LXD servers

You can copy or move custom storage volumes between different LXD 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_volume_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.
Move instance storage volumes to another pool

To move an instance storage volume to another storage pool, make sure the instance is stopped. Then use the following command to move the instance to a different pool:

```
lxc move <instance_name> --storage <target_pool_name>
```

Related topics

Explanation:

• About storage pools, volumes and buckets

Reference:

• Storage drivers

2.2.11 Troubleshooting

The following how-to guides cover common operations related to troubleshooting:

How to debug LXD

For information on debugging instance issues, see How to troubleshoot failing instances.

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 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* for available API.

**REST API through HTTPS**

*HTTPS connection to LXD* requires valid client certificate that is generated on first `lxc remote add`. 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:

<table>
<thead>
<tr>
<th>Certificate purposes:</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL client : Yes</td>
</tr>
</tbody>
</table>

**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:

```
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 the LXD database

The files of the global database are stored under the .database/global sub-directory 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 or forum 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.

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.
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 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 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.

When I do a lxc remote add, it asks for a password or token?

To be able to access the remote API, clients must authenticate with the LXD server. Depending on how the remote server is configured, you must provide either a trust token issued by the server or specify a trust password (if core.trust_password is set).

See Authenticate with the LXD server for instructions on how to authenticate using a trust token (the recommended way), and Remote API authentication for information about other authentication methods.

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 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:

```
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 call. This depends on the kernel and file system supporting either idmapped mounts or shiftfs (see lxc info).
- Add a raw.idmap entry (see Idmaps for user namespace).
- 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` property of the container to `true`:

```
lxc config set <container> security.nesting true
```

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` 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` 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` for all options, and *How to debug LXD* 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>`). 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` (see How can I monitor what LXD is doing?), 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 and Private Internet Access VPN, 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 for instructions for Mullvad VPN.

2.3 Explanation

The explanatory guides in this section introduce you to the concepts used in LXD and help you understand how things fit together.

2.3.1 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 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
- OpenID Connect authentication
- Candid-based authentication

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.

It’s possible to force LXD to accept TLS 1.2 by setting the `LXD_INSECURE_TLS` environment variable on both client and server. However this isn’t a supported setup and should only ever be used when forced to use an outdated corporate proxy.

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**

You can obtain the list of TLS certificates trusted by a LXD server with `lxc config trust list`.

Trusted clients can be added in either of the following ways:

- Adding trusted certificates to the server
- Adding client certificates using a trust password
- Adding client certificates using tokens

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`, 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 or the trust password. If the provided token or trust password matches, the client certificate is added to the server’s trust store and the connection is granted. Otherwise, the connection is rejected.

It is possible to restrict a TLS client’s access to LXD via TLS authorization. To revoke trust to a client, remove its certificate from the server with `lxc config trust remove <fingerprint>`.
Adding trusted certificates to the server

The preferred way to add trusted clients is to directly add their certificates to the trust store on the server. To do so, copy the client certificate to the server and register it using `lxc config trust add <file>`.

Adding client certificates using a trust password

To allow establishing a new trust relationship from the client side, you must set a trust password (`core.trust_password`) for the server. Clients can then add their own certificate to the server’s trust store by providing the trust password when prompted.

In a production setup, unset `core.trust_password` after all clients have been added. This prevents brute-force attacks trying to guess the password.

Adding client certificates using tokens

You can also add new clients by using tokens. This is a safer way than using the trust password, because tokens expire after a configurable time (`core.remote_token_expiry`) or once they’ve been used.

To use this method, generate a token for each client by calling `lxc config trust add`, which will prompt for the client name. The clients can then add their certificates to the server’s trust store by providing the generated token when prompted for the trust password.

**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 admin password, specify the generated 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.

Alternatively, the clients can provide the token directly when adding the remote: `lxc remote add <name> <token>`.

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.

To enable PKI mode, complete the following steps:

1. Add the CA (Certificate authority) 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).
2. Place the certificates issued by the CA on the clients and the server, replacing the automatically generated ones.
3. Restart the server.
In that mode, any connection to a LXD daemon will be done using the pre-seeded CA certificate. If the server certificate isn’t signed by the CA, the connection will simply go through the normal authentication mechanism. If the server certificate is valid and signed by the CA, then the connection continues without prompting the user for the certificate.

Note that the generated certificates are not automatically trusted. You must still add them to the server in one of the ways described in Trusted TLS clients.

**OpenID Connect authentication**

LXD supports using OpenID Connect to authenticate users through an OIDC (OpenID Connect) Identity Provider. To configure LXD to use OIDC authentication, set the `oidc.*` server configuration options. Your OIDC provider must be configured to enable the Device Authorization Grant type.

To add a remote pointing to a LXD server configured with OIDC authentication, run `lxc remote add <remote_name> <remote_address>`. You are then prompted to authenticate through your web browser, where you must confirm the device code that LXD uses. The LXD client then retrieves and stores the access and refresh tokens and provides those to LXD for all interactions.

**Important:** Any user that authenticates through the configured OIDC Identity Provider gets full access to LXD. To restrict user access, you must also configure Authorization. Currently, the only authorization method that is compatible with OIDC is Open Fine-Grained Authorization (OpenFGA).

**Candid-based authentication**

You can configure LXD to use Candid authentication by setting the `candid.*` server configuration options. In this case, clients that try to authenticate with the server must get a Discharge token from the authentication server specified by the `candid.api_url` option.

The authentication server certificate must be trusted by the LXD server.

To add a remote pointing to a LXD server configured with Candid/Macaroon authentication, run `lxc remote add REMOTE_ENDPOINT --auth-type=candid`. To verify the user, the client will prompt for the credentials required by the authentication server. If the authentication is successful, the client will connect to the LXD server and present the token received from the authentication server. The LXD server verifies the token, thus authenticating the request. The token is stored as cookie and is presented by the client at each request to LXD.

For instructions on how to set up Candid-based authentication, see the Candid authentication for LXD tutorial.

**Important:** Any user that authenticates via Candid gets full access to LXD. To restrict user access, you must also configure Authorization. Candid is compatible with Role Based Access Control (RBAC) and Open Fine-Grained Authorization (OpenFGA).
TLS server certificate

LXD supports issuing server certificates using ACME (Automatic Certificate Management Environment) services, for example, Let’s Encrypt.

To enable this feature, set the following server configuration:

- `acme.domain`: The domain for which the certificate should be issued.
- `acme.email`: The email address used for the account of the ACME service.
- `acme.agree_tos`: Must be set to `true` to agree to the ACME service’s terms of service.
- `acme.ca_url`: 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.

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/`.

```plaintext
# 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
```

(continues on next page)
bind :80
mode http

# Backend selection
tcp-request inspect-delay 5s

# 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:

• About security

How-to guides:

• How to expose LXD to the network

2.3.2 Authorization

When interacting with LXD over the Unix socket, clients have full access to the LXD API. However, it is possible to restrict user access to the LXD API when communicating via remote HTTPS (see How to expose LXD to the network for instructions). There are three supported authorization methods:

• TLS authorization
• Role Based Access Control (RBAC)
• Open Fine-Grained Authorization (OpenFGA)

TLS authorization

LXD natively supports restricting Trusted TLS clients to one or more projects. When a client certificate is restricted, 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>. 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.

This authorization method is always used if a client authenticates with TLS, regardless of whether another authorization method is configured.

Role Based Access Control (RBAC)

LXD supports integrating with the Canonical RBAC service, which is included in the Ubuntu Pro subscription. RBAC (Role Based Access Control) can be used to limit what an API client is allowed to do on LXD. This authorization method can only be used with Candid-based authentication.

In such a setup, authentication happens through Candid, while the RBAC service maintains roles to user/group relationships. Roles can be assigned to individual projects, to all projects or to the entire LXD instance.

The meaning of the roles when applied to a project is as follows:

• auditor: Read-only access to the project
• **user**: Ability to do normal life cycle actions (start, stop, ...), execute commands in the instances, attach to console, manage snapshots, ...

• **operator**: All of the above + the ability to create, re-configure and delete instances and images

• **admin**: All of the above + the ability to reconfigure the project itself

To enable RBAC for your LXD server, set the `rbac.*` server configuration options, which are a superset of the `candid.*` ones and allow for LXD to integrate with the RBAC service.

---

**Important**: In an unrestricted project, only the auditor and the user roles are suitable for users that you wouldn’t trust with root access to the host.

In a **restricted project**, the operator role is safe to use as well if configured appropriately.

---

**Open Fine-Grained Authorization (OpenFGA)**

LXD supports integrating with OpenFGA (Open Fine-Grained Authorization). This authorization method is highly granular. For example, it can be used to restrict user access to a single instance. OpenFGA authorization is compatible with **Candid-based authentication** and **OpenID Connect authentication**.

To use OpenFGA for authorization, you must configure and run an OpenFGA server yourself. To enable this authorization method in LXD, set the `openfga.*` server configuration options. LXD will connect to the OpenFGA server, write the OpenFGA model, and query this server for authorization for all subsequent requests.

**OpenFGA model**

With OpenFGA, access to a particular API resource is determined by the user’s relationship to it. These relationships are determined by an OpenFGA authorization model. The LXD OpenFGA authorization model describes API resources in terms of their relationship to other resources, and a relationship a user or group might have with that resource. Some convenient relations have also been built into the model:

- `server -> admin`: Full access to LXD.
- `server -> operator`: Full access to LXD, without edit access on server configuration, certificates, or storage pools.
- `server -> viewer`: Can view all server level configuration but cannot edit. Cannot view projects or their contents.
- `project -> manager`: Full access to a single project, including edit access.
- `project -> operator`: Full access to a single project, without edit access.
- `project -> viewer`: View access for a single project.
- `instance -> manager`: Full access to a single instance, including edit access.
- `instance -> operator`: Full access to a single instance, without edit access.
- `instance -> user`: View access to a single instance, plus permissions for exec, console, and file APIs.
- `instance -> viewer`: View access to a single instance.

---

**Important**: Users that you do not trust with root access to the host should not be granted the following relations:

- `server -> admin`
- `server -> operator`
The remaining relations may be granted. However, you must apply appropriate Project restrictions.

The full LXD OpenFGA authorization model is defined in `lxd/auth/driver_openfga_model.openfga`:

```
model
  schema 1.1
  type user
    relations
      define member: [user]
  type server
    relations
      define admin: [user, group#member] or admin
      define operator: [user, group#member] or operator
      define viewer: [user, group#member] or viewer
      define user: [user:*]
      define can_edit: admin
      define can_view: user
      define can_create_storage_pools: [user, group#member] or admin
      define can_create_projects: [user, group#member] or operator
      define can_create_certificates: [user, group#member] or admin
      define can_view_resources: [user, group#member] or viewer
      define can_view_metrics: [user, group#member] or viewer
      define can_override_cluster_target_restriction: [user, group#member] or admin
      define can_view_privileged_events: [user, group#member] or admin
  type certificate
    relations
      define server: [server]
      define can_edit: [user, group#member] or admin from server
      define can_view: user from server
  type storage_pool
    relations
      define server: [server]
      define can_edit: [user, group#member] or admin from server
      define can_view: user from server
  type project
    relations
      define server: [server]
      define manager: [user, group#member] or operator from server
      define operator: [user, group#member] or manager or operator from server
      define viewer: [user, group#member] or operator
      define can_edit: manager
```

(continues on next page)
define can_view: viewer
define can_create_images: [user, group#member] or operator or operator from server
define can_create_image_aliases: [user, group#member] or operator or operator from server
define can_create_instances: [user, group#member] or operator or operator from server
define can_create_networks: [user, group#member] or operator or operator from server
define can_create_network_acls: [user, group#member] or operator or operator from server
define can_create_network_zones: [user, group#member] or operator or operator from server
define can_create_profiles: [user, group#member] or operator or operator from server
define can_create_storage_volumes: [user, group#member] or operator or operator from server
define can_create_storage_buckets: [user, group#member] or operator or operator from server
define can_view_operations: [user, group#member] or viewer
define can_view_events: [user, group#member] or viewer
type image
relations
  define project: [project]
  define can_edit: [user, group#member] or operator from project
  define can_view: [user, group#member] or can_edit or viewer from project
type image_alias
relations
  define project: [project]
  define can_edit: [user, group#member] or operator from project
  define can_view: [user, group#member] or can_edit or viewer from project
type instance
relations
  define project: [project]
  define manager: [user, group#member]
  define operator: [user, group#member] or manager
  define user: [user, group#member] or operator
  define viewer: [user, group#member] or operator
  define can_edit: manager or operator from project
  define can_view: user or viewer or viewer from project
  define can_update_state: [user, group#member] or operator or operator from project
  define can_manage_snapshots: [user, group#member] or operator or operator from project
  define can_manage_backups: [user, group#member] or operator or operator from project
  define can_connect_sftp: [user, group#member] or user or operator from project
  define can_access_files: [user, group#member] or user or operator from project
  define can_access_console: [user, group#member] or user or operator from project
  define can_exec: [user, group#member] or user or operator from project
  define can_connect_to: [user, group#member] or user or operator from project
type network
relations
  define project: [project]
  define can_edit: [user, group#member] or operator from project
  define can_view: [user, group#member] or can_edit or viewer from project
type network_acl
relations
  define project: [project]
2.3.3 About clustering

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` client or the REST API.

This feature was introduced as part of the `clustering` API extension and is available since LXD 3.0.

**Tip:** If you want to quickly set up a basic LXD cluster, check out MicroCloud.

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, which is a Dqlite 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* 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 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.

<table>
<thead>
<tr>
<th>Role</th>
<th>Automatic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>yes</td>
<td>Voting member of the distributed database</td>
</tr>
<tr>
<td>database-leader</td>
<td>yes</td>
<td>Current leader of the distributed database</td>
</tr>
<tr>
<td>database-standby</td>
<td>yes</td>
<td>Stand-by (non-voting) member of the distributed database</td>
</tr>
<tr>
<td>event-hub</td>
<td>no</td>
<td>Exchange point (hub) for the internal LXD events (requires at least two)</td>
</tr>
<tr>
<td>ovn-chassis</td>
<td>no</td>
<td>Uplink gateway candidate for OVN networks</td>
</tr>
</tbody>
</table>

The default number of voter members (cluster.max_voters) is three. The default number of stand-by members (cluster.max_standby) 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* 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.

You can tweak the amount of seconds after which a non-responding member is considered offline by setting the cluster.offline_threshold configuration. The default value is 20 seconds. The minimum value is 10 seconds.

To automatically evacuate instances from an offline member, set the cluster.healing_threshold configuration to a non-zero value.

See *How to recover a cluster* for more information.
Canonical LXD

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>` 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 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* and *How to configure networks for a cluster* 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` 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* and *Launch an instance on a specific cluster member* 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` 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 (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:

```python
instance_placement(request, candidate_members):
```

- `request` is an object that contains an expanded representation of `scriptlet.InstancePlacement`. This request includes `project` and `reason` fields. The reason can be `new`, `evacuation` or `relocation`.
- `candidate_members` is a list of cluster member objects representing `api.ClusterMember` entries.

For example:

```python
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)
```

(continues on next page)
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` 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`. `member_name` is the name of the cluster member to get the state for.
- `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`. `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`.

**Note:** Field names in the object types are equivalent to the JSON field names in the associated Go types.
2.3.4 About containers and VMs

LXD provides support for two different types of instances: 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 light-weight 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.
2.3.5 About 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 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.
- The image's expiry date (one of the image properties; see Edit image properties 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 was set to true (the default) at download time.
- If the image was copied from a remote server using the lxc image copy 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.

On startup and after every images.auto_update_interval (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:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requirements.secureboot</td>
<td>string</td>
<td>-</td>
<td>If set to false, indicates that the image cannot boot under secure boot.</td>
</tr>
<tr>
<td>requirements.cgroup</td>
<td>string</td>
<td>-</td>
<td>If set to v1, indicates that the image requires the host to run cgroup v1.</td>
</tr>
</tbody>
</table>

Related topics

How-to guides:
- Images

Reference:
- Image format
- Remote image servers

2.3.6 About instances

LXD supports the following types of instances:

Containers

Containers are the default type for instances. They are currently the most complete implementation of LXD instances and support more features than virtual machines.

Containers 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.

LXD uses qemu to provide the VM functionality.

Note: Currently, virtual machines support fewer features than containers, but the plan is to support the same set of features for both instance types in the future.

To see which features are available for virtual machines, check the condition column in the Instance options documentation.
2.3.7 About 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 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 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 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.

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.
LXD client
The \texttt{lxc} command is a command-line client for LXD, which you can use to interact with the LXD daemon. You use the \texttt{lxc} command to manage your instances, the server settings, and overall the entities you create in LXD. See \texttt{lxc \--help} for an overview of all available subcommands.

The \texttt{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.

2.3.8 About networking

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:
  \begin{verbatim}
  lxc profile show default
  \end{verbatim}
  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:
  \begin{verbatim}
  lxc config device add <instance_name> <device_name> nic nictype=<nic_type> ...
  \end{verbatim}
  See \texttt{Type: nic} 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:
  \begin{verbatim}
  lxc config device add <instance_name> eth0 nic nictype=bridged parent=br0
  \end{verbatim}

- Create a managed network 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:
  \begin{verbatim}
  lxc network attach <network_name> <instance_name> <device_name>
  \end{verbatim}
  See \texttt{Attach a network to an instance} 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 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**

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**

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](http://www.ovn.org) 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. 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**

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**

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**

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.

**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*, possibly in connection with the Ubuntu Fan.
- If you are running LXD in your own private cloud, use an *OVN network*.

**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 `nic_type`.

**Related topics**

How-to guides:

- *Networking*

Reference:

- *Networks*

**2.3.9 About 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* 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* for lists of available metrics.

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* for more information about metrics collection.

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* 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* for instructions.

Related topics

How-to guides:

- *How to benchmark performance*
- *How to increase the network bandwidth*
- *How to monitor metrics*

Reference:

- *Provided metrics*
2.3.10 About 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 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 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, 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 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.
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*. 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*). 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.

Authentication methods for projects

There are different ways of authentication that you can use to confine projects to specific users:

**Client certificates**

You can restrict the *TLS client certificates* to allow access to specific projects only. The projects must exist before you can restrict access to them. A client that connects using a restricted certificate can see only the project or projects that the client has been granted access to.

**RBAC authentication**

If you use *Role Based Access Control (RBAC)*, you can assign roles to users or groups both on a global and on a project basis. The roles define what a user is allowed to do within the project. This way, you can configure which users are allowed to see, use, or manage each project.

**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, LXD 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 *How to confine projects to specific users* for instructions on how to enable and configure the different authentication methods.

Related topics

How-to guides:

- *Projects*

Reference:

- *Project configuration*
2.3.11 About 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 monthly feature releases).
- Restrict access to the LXD daemon and the remote API.
- Do not use privileged containers unless required. If you use privileged containers, put appropriate security measures in place. See the LXD security page for more information.
- Configure your network interfaces to be secure.

See the following sections for detailed information.

If you discover a security issue, see the LXD security policy for information on how to report the issue.

Supported versions

Never use unsupported LXD versions in a production environment.

LXD has two types of releases:

- Monthly 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 monthly 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` configuration option, you can expose the same API over the network on a TLS socket. See *How to expose LXD to the network* 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* for details.

In a production setup, you should set `core.https_address` 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.

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.

If data sharing between containers isn’t needed, you can enable `security.idmap.isolated`, which will use non-overlapping UID/GID maps for each container, preventing potential DoS (Denial of Service) attacks on other containers.

LXD can also run *privileged* containers. Note, however, that those aren’t 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.

More details on container security and the kernel features we use can be found on the LXC security page.

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:

```
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 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:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>security.mac_filtering</td>
<td>bool</td>
<td>false</td>
<td>no</td>
<td>Prevent the instance from spoofing another instance’s MAC address</td>
</tr>
<tr>
<td>security.ipv4_filtering</td>
<td>bool</td>
<td>false</td>
<td>no</td>
<td>Prevent the instance from spoofing another instance’s IPv4 address (enables mac_filtering)</td>
</tr>
<tr>
<td>security.ipv6_filtering</td>
<td>bool</td>
<td>false</td>
<td>no</td>
<td>Prevent the instance from spoofing another instance’s IPv6 address (enables mac_filtering)</td>
</tr>
</tbody>
</table>

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*

Explanation:

- *Remote API authentication*

2.3.12 About 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) protocol. Like storage volumes, storage buckets are part of a storage pool.

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*).

Each storage pool uses a storage driver. The following storage drivers are supported:

- *Directory - dir*
- *Btrfs - btrfs*
- *LVM - lvmlv*
- *ZFS - zfs*
- *Ceph RBD - ceph*
- *CephFS - cephfs*
- *Ceph Object - cephobject*

See the following how-to guides for additional information:

- *How to manage storage pools*
- *How to create an instance in a specific storage pool*
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.

<table>
<thead>
<tr>
<th>Storage location</th>
<th>Directory</th>
<th>Btrfs</th>
<th>LVM</th>
<th>ZFS</th>
<th>Ceph (all)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared with the host</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Dedicated disk/partition</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Loop disk</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Remote storage</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
</tbody>
</table>

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 though; see Resize a storage pool.

Remote storage

The ceph, cephfs and cephobject drivers store the data in a completely independent Ceph storage cluster that must be set up separately.
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>`
- Through a profile: `lxc profile device add <profile_name> root disk path=/ pool=<storage_pool>` and `lxc launch <image> <instance_name> --profile <profile_name>`
- 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*
- *How to move or copy storage volumes*
- *How to back up custom storage volumes*

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 it is destroyed when the instance is deleted.

This storage volume is created in the storage pool that is specified in the profile used when launching the instance (or the default profile, if no profile is specified). The storage pool can be explicitly specified by providing the `--storage` flag to the launch command.

**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* that supports optimized image storage.
You can add one or more custom storage volumes to hold data that you want to store separately from your
instances. Custom storage volumes 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. They should not be
shared between instances, because simultaneous access can lead to data corruption.

**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 import.

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 server setting.

See the following how-to guide for additional information:

- How to manage storage buckets and keys
2.3.13 About the LXD 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, 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.

**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 for more information.

**Backup**

See Back up the database for instructions on how to back up the contents of the LXD database.

2.4 Reference

The reference material in this section provides technical descriptions of LXD.
2.4.1 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.

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Notes</th>
<th>Personalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>i686</td>
<td>32bit Intel x86</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>x86_64</td>
<td>64bit Intel x86</td>
<td>x86</td>
</tr>
<tr>
<td>3</td>
<td>armv7l</td>
<td>32bit ARMv7 little-endian</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>aarch64</td>
<td>64bit ARMv8 little-endian</td>
<td>armv7 (optional)</td>
</tr>
<tr>
<td>5</td>
<td>ppc</td>
<td>32bit PowerPC big-endian</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ppc64</td>
<td>64bit PowerPC big-endian</td>
<td>powerpc</td>
</tr>
<tr>
<td>7</td>
<td>ppc64le</td>
<td>64bit PowerPC little-endian</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>s390x</td>
<td>64bit ESA/390 big-endian</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>mips</td>
<td>32bit MIPS</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>mips64</td>
<td>64bit MIPS</td>
<td>mips</td>
</tr>
<tr>
<td>11</td>
<td>riscv32</td>
<td>32bit RISC-V little-endian</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>riscv64</td>
<td>64bit RISC-V little-endian</td>
<td></td>
</tr>
</tbody>
</table>

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 armv7. If useful to the user, the exact userspace ABI may be set as an image and container property, allowing easy query.

2.4.2 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:   

<table>
<thead>
<tr>
<th>Key:</th>
<th>scheduler.instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>all</td>
</tr>
</tbody>
</table>

Possible values are all, manual, and group. See *Automatic placement of instances* for more information.
user.* Free form user key/value storage

<table>
<thead>
<tr>
<th>Key</th>
<th>user.*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
</tbody>
</table>

User keys can be used in search.

**Related topics**

How-to guides:

- *Clustering*

Explanation:

- *About clustering*

### 2.4.3 Index

### 2.4.4 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)

**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* 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
• /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

Explanation:

- About instances

2.4.5 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 22.04 LTS Intel 64bit
  os: Ubuntu
  release: jammy 22.04
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) 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:

```yaml
templates:
  /etc/hosts:
    when:
      - create
      - rename
      template: hosts.tpl
    properties:
      foo: bar
  /etc/hostname:
    when:
      - start
      template: hostname.tpl
  /etc/network/interfaces:
    when:
      - create
```

(continues on next page)
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 format.

They always receive the following context:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trigger</td>
<td>string</td>
<td>Name of the event that triggered the template</td>
</tr>
<tr>
<td>path</td>
<td>string</td>
<td>Path of the file that uses the template</td>
</tr>
<tr>
<td>instance</td>
<td>map[string]string</td>
<td>Key/value map of instance properties (name, architecture, privileged and ephemeral)</td>
</tr>
<tr>
<td>config</td>
<td>map[string]string</td>
<td>Key/value map of the instance’s configuration</td>
</tr>
<tr>
<td>devices</td>
<td>map[string]map[string]string</td>
<td>Key/value map of the devices assigned to the instance</td>
</tr>
<tr>
<td>properties</td>
<td>map[string]string</td>
<td>Key/value map of the template properties specified in metadata.yaml</td>
</tr>
</tbody>
</table>

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 meta-data, 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.

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

Explanation:

- About images

2.4.6 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 or editing the full instance configuration.

In the YAML configuration, properties are on the top level.

See Instance properties 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 and lxc config unset commands.

In the YAML configuration, options are located under the config entry.

See Instance options for a reference of available instance options, and Configure instance options 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` 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* for a reference of available devices and the corresponding instance device options, and *Configure devices* for instructions on how to add and configure instance devices.

**Devices**

Devices are attached to an instance (see *Configure devices*) or to a profile (see *Edit a profile*).

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:

<table>
<thead>
<tr>
<th>ID (database)</th>
<th>Name</th>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>none</td>
<td>-</td>
<td>Inheritance blocker</td>
</tr>
<tr>
<td>1</td>
<td>nic</td>
<td>-</td>
<td>Network interface</td>
</tr>
<tr>
<td>2</td>
<td>disk</td>
<td>-</td>
<td>Mount point inside the instance</td>
</tr>
<tr>
<td>3</td>
<td>unix-char</td>
<td>container</td>
<td>Unix character device</td>
</tr>
<tr>
<td>4</td>
<td>unix-block</td>
<td>container</td>
<td>Unix block device</td>
</tr>
<tr>
<td>5</td>
<td>usb</td>
<td>-</td>
<td>USB device</td>
</tr>
<tr>
<td>6</td>
<td>gpu</td>
<td>-</td>
<td>GPU device</td>
</tr>
<tr>
<td>7</td>
<td>infiniband</td>
<td>container</td>
<td>InfiniBand device</td>
</tr>
<tr>
<td>8</td>
<td>proxy</td>
<td>container</td>
<td>Proxy device</td>
</tr>
<tr>
<td>9</td>
<td>unix-hotplug</td>
<td>container</td>
<td>Unix hotplug device</td>
</tr>
<tr>
<td>10</td>
<td>tpm</td>
<td>-</td>
<td>TPM device</td>
</tr>
<tr>
<td>11</td>
<td>pci</td>
<td>VM</td>
<td>PCI device</td>
</tr>
</tbody>
</table>

Each instance comes with a set of *Standard devices*.

**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:
<table>
<thead>
<tr>
<th>Device</th>
<th>Type of device</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/null</td>
<td>Character device</td>
</tr>
<tr>
<td>/dev/zero</td>
<td>Character device</td>
</tr>
<tr>
<td>/dev/full</td>
<td>Character device</td>
</tr>
<tr>
<td>/dev/console</td>
<td>Character device</td>
</tr>
<tr>
<td>/dev/tty</td>
<td>Character device</td>
</tr>
<tr>
<td>/dev/random</td>
<td>Character device</td>
</tr>
<tr>
<td>/dev/urandom</td>
<td>Character device</td>
</tr>
<tr>
<td>/dev/net/tun</td>
<td>Character device</td>
</tr>
<tr>
<td>/dev/fuse</td>
<td>Character device</td>
</tr>
<tr>
<td>lo</td>
<td>Network interface</td>
</tr>
</tbody>
</table>

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.

**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. 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” column of the NIC-specific tables of device options. You cannot customize these options directly for the NIC if you’re using the network method.

See About networking for more information.

**Available NIC types**

The following NICs can be added using the nictype or network options:

- **bridged**: Uses an existing bridge on the host and creates a virtual device pair to connect the host bridge to the instance.
- **macvlan**: Sets up a new network device based on an existing one, but using a different MAC address.
- **sriov**: Passes a virtual function of an SR-IOV-enabled physical network device into the instance.
- **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.

The following NICs can be added using only the network option:

- **ovn**: 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**: Sets up a new network device based on an existing one, using the same MAC address but a different IP.
- **p2p**: Creates a virtual device pair, putting one side in the instance and leaving the other side on the host.
- **routed**: 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 tables in the following sections.

**nictype: bridged**

**Note**: You can select this NIC type through the nictype option or the network option (see Bridge network 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:
<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Managed</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boot.priority</td>
<td>integer</td>
<td>-</td>
<td>no</td>
<td>Boot priority for VMs (higher value boots first)</td>
</tr>
<tr>
<td>host_name</td>
<td>string</td>
<td>randomly assigned</td>
<td>no</td>
<td>The name of the interface inside the host</td>
</tr>
<tr>
<td>hwaddr</td>
<td>string</td>
<td>randomly assigned</td>
<td>no</td>
<td>The MAC address of the new interface</td>
</tr>
<tr>
<td>ipv4.address</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>An IPv4 address to assign to the instance through DHCP (can be none to restrict all IPv4 traffic when security.ipv4_filtering is set)</td>
</tr>
<tr>
<td>ipv4.routes</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>Comma-delimited list of IPv4 static routes to add on host to NIC</td>
</tr>
<tr>
<td>ipv4.routes.externa</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>Comma-delimited list of IPv4 static routes to route to the NIC and publish on uplink network (BGP)</td>
</tr>
<tr>
<td>ipv6.address</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>An IPv6 address to assign to the instance through DHCP (can be none to restrict all IPv6 traffic when security.ipv6_filtering is set)</td>
</tr>
<tr>
<td>ipv6.routes</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>Comma-delimited list of IPv6 static routes to add on host to NIC</td>
</tr>
<tr>
<td>ipv6.routes.externa</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>Comma-delimited list of IPv6 static routes to route to the NIC and publish on uplink network (BGP)</td>
</tr>
<tr>
<td>limits.egress</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>I/O limit in bit/s for outgoing traffic (various suffixes supported, see Units for storage and network limits)</td>
</tr>
<tr>
<td>limits.ingress</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>I/O limit in bit/s for incoming traffic (various suffixes supported, see Units for storage and network limits)</td>
</tr>
<tr>
<td>limits.max</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>I/O limit in bit/s for both incoming and outgoing traffic (same as setting both limits.ingress and limits.egress)</td>
</tr>
<tr>
<td>limits.priorit</td>
<td>integer</td>
<td>-</td>
<td>no</td>
<td>The skb-&gt;priority value (32-bit unsigned integer) for outgoing traffic, to be used by the kernel queuing discipline (qdisc) to prioritize network packets (The effect of this value depends on the particular qdisc implementation, for example, SKBFIFO or QFQ. Consult the kernel qdisc documentation before setting this value.)</td>
</tr>
<tr>
<td>maas.subnet.ipv4</td>
<td>string</td>
<td>-</td>
<td>yes</td>
<td>MAAS IPv4 subnet to register the instance in</td>
</tr>
<tr>
<td>maas.subnet.ipv6</td>
<td>string</td>
<td>-</td>
<td>yes</td>
<td>MAAS IPv6 subnet to register the instance in</td>
</tr>
<tr>
<td>mtu</td>
<td>integer</td>
<td>parent</td>
<td>yes</td>
<td>The MTU of the new interface</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>kernel assigned</td>
<td>no</td>
<td>The name of the interface inside the instance</td>
</tr>
<tr>
<td>network</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>The managed network to link the device to (instead of specifying the nictype directly)</td>
</tr>
<tr>
<td>parent</td>
<td>string</td>
<td>-</td>
<td>yes</td>
<td>The name of the host device (required if specifying the nictype directly)</td>
</tr>
<tr>
<td>queue</td>
<td>integer</td>
<td>-</td>
<td>no</td>
<td>The transmit queue length for the NIC</td>
</tr>
<tr>
<td>security</td>
<td>boolean</td>
<td>false</td>
<td>no</td>
<td>Prevent the instance from spoofing another instance’s IPv4 address (enables security.mac_filtering)</td>
</tr>
<tr>
<td>security.ipv4_filtering</td>
<td>boolean</td>
<td>false</td>
<td>no</td>
<td>Prevent the instance from spoofing another instance’s IPv4 address (enables security.mac_filtering)</td>
</tr>
<tr>
<td>security.ipv6_filtering</td>
<td>boolean</td>
<td>false</td>
<td>no</td>
<td>Prevent the instance from spoofing another instance’s IPv6 address (enables security.mac_filtering)</td>
</tr>
</tbody>
</table>
**nictype: macvlan**

**Note:** You can select this NIC type through the nictype option or the network option (see *Macvlan network* 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:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Managed</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boot.priority</td>
<td>integer</td>
<td>-</td>
<td>no</td>
<td>Boot priority for VMs (higher value boots first)</td>
</tr>
<tr>
<td>gvrp</td>
<td>bool</td>
<td>false</td>
<td>no</td>
<td>Register VLAN using GARP VLAN Registration Protocol</td>
</tr>
<tr>
<td>hwaddr</td>
<td>string</td>
<td>randomly assigned</td>
<td>no</td>
<td>The MAC address of the new interface</td>
</tr>
<tr>
<td>maas.subnet.ipv4</td>
<td>string</td>
<td>-</td>
<td>yes</td>
<td>MAAS IPv4 subnet to register the instance in</td>
</tr>
<tr>
<td>maas.subnet.ipv6</td>
<td>string</td>
<td>-</td>
<td>yes</td>
<td>MAAS IPv6 subnet to register the instance in</td>
</tr>
<tr>
<td>mtu</td>
<td>integer</td>
<td>parent MTU</td>
<td>yes</td>
<td>The MTU of the new interface</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>kernel assigned</td>
<td>no</td>
<td>The name of the interface inside the instance</td>
</tr>
<tr>
<td>network</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>The managed network to link the device to (instead of specifying the nictype directly)</td>
</tr>
<tr>
<td>parent</td>
<td>string</td>
<td>-</td>
<td>yes</td>
<td>The name of the host device (required if specifying the nictype directly)</td>
</tr>
<tr>
<td>vlan</td>
<td>integer</td>
<td>-</td>
<td>no</td>
<td>The VLAN ID to attach to</td>
</tr>
</tbody>
</table>

**nictype: sriov**

**Note:** You can select this NIC type through the nictype option or the network option (see *SR-IOV network* 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:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Managed</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boot.priority</td>
<td>integer</td>
<td>-</td>
<td>no</td>
<td>Boot priority for VMs (higher value boots first)</td>
</tr>
<tr>
<td>hwaddr</td>
<td>string</td>
<td>randomly assigned</td>
<td>no</td>
<td>The MAC address of the new interface</td>
</tr>
<tr>
<td>maas.subnet.ipv4</td>
<td>string</td>
<td>-</td>
<td>yes</td>
<td>MAAS IPv4 subnet to register the instance in</td>
</tr>
<tr>
<td>maas.subnet.ipv6</td>
<td>string</td>
<td>-</td>
<td>yes</td>
<td>MAAS IPv6 subnet to register the instance in</td>
</tr>
<tr>
<td>mtu</td>
<td>integer</td>
<td>kernel assigned</td>
<td>yes</td>
<td>The MTU of the new interface</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>kernel assigned</td>
<td>no</td>
<td>The name of the interface inside the instance</td>
</tr>
<tr>
<td>network</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>The managed network to link the device to (instead of specifying the <code>nictype</code> directly)</td>
</tr>
<tr>
<td>parent</td>
<td>string</td>
<td>-</td>
<td>yes</td>
<td>The name of the host device (required if specifying the <code>nictype</code> directly)</td>
</tr>
<tr>
<td>security.mac_filtering</td>
<td>bool</td>
<td>false</td>
<td>no</td>
<td>Prevent the instance from spoofing another instance’s MAC address</td>
</tr>
<tr>
<td>vlan</td>
<td>integer</td>
<td>-</td>
<td>no</td>
<td>The VLAN ID to attach to</td>
</tr>
</tbody>
</table>
nictype: ovn

Note: You can select this NIC type only through the network option (see OVN network 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.

```bash
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
device 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. Setup OVS by enabling hardware offload and adding the PF NIC to the integration bridge (normally called br-int):

```bash
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:

```bash
modprobe vhost_vdpa && vdpa mgmtdev show
```
## Device options

NIC devices of type ovn have the following device options:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Managed</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acceleration</td>
<td>string</td>
<td>none</td>
<td>no</td>
<td>Enable hardware offloading (either none, sriov or vdpa, see SR-IOV hardware acceleration)</td>
</tr>
<tr>
<td>boot.priority</td>
<td>integer</td>
<td>-</td>
<td>no</td>
<td>Boot priority for VMs (higher value boots first)</td>
</tr>
<tr>
<td>host_name</td>
<td>string</td>
<td>randomly assigned</td>
<td>no</td>
<td>The name of the interface inside the host</td>
</tr>
<tr>
<td>hwaddr</td>
<td>string</td>
<td>randomly assigned</td>
<td>no</td>
<td>The MAC address of the new interface</td>
</tr>
<tr>
<td>ipv4.address</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>An IPv4 address to assign to the instance through DHCP</td>
</tr>
<tr>
<td>ipv4.routes</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>Comma-delimited list of IPv4 static routes to route to the NIC</td>
</tr>
<tr>
<td>ipv4.routes.external</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>Comma-delimited list of IPv4 static routes to route to the NIC and publish on uplink network</td>
</tr>
<tr>
<td>ipv6.address</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>An IPv6 address to assign to the instance through DHCP</td>
</tr>
<tr>
<td>ipv6.routes</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>Comma-delimited list of IPv6 static routes to route to the NIC</td>
</tr>
<tr>
<td>ipv6.routes.external</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>Comma-delimited list of IPv6 static routes to route to the NIC and publish on uplink network</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>kernel assigned</td>
<td>no</td>
<td>The name of the interface inside the instance</td>
</tr>
<tr>
<td>nested</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>The parent NIC name to nest this NIC under (see also vlan)</td>
</tr>
<tr>
<td>network.acls</td>
<td>string</td>
<td>-</td>
<td>yes</td>
<td>The managed network to link the device to (required)</td>
</tr>
<tr>
<td>security.acls</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>Comma-separated list of network ACLs to apply</td>
</tr>
<tr>
<td>security.acls.default.egress.action</td>
<td>string</td>
<td>reject</td>
<td>no</td>
<td>Action to use for egress traffic that doesn’t match any ACL rule</td>
</tr>
<tr>
<td>security.acls.default.egress.logged</td>
<td>bool</td>
<td>false</td>
<td>no</td>
<td>Whether to log egress traffic that doesn’t match any ACL rule</td>
</tr>
<tr>
<td>security.acls.default.ingress.action</td>
<td>string</td>
<td>reject</td>
<td>no</td>
<td>Action to use for ingress traffic that doesn’t match any ACL rule</td>
</tr>
<tr>
<td>security.acls.default.ingress.logged</td>
<td>bool</td>
<td>false</td>
<td>no</td>
<td>Whether to log ingress traffic that doesn’t match any ACL rule</td>
</tr>
<tr>
<td>vlan</td>
<td>integer</td>
<td>-</td>
<td>no</td>
<td>The VLAN ID to use when nesting (see also nested)</td>
</tr>
</tbody>
</table>
nictype: physical

**Note:**

- You can select this NIC type through the `nictype` option or the `network` option (see *Physical network* 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:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Managed</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boot.priority</td>
<td>integer</td>
<td>-</td>
<td>no</td>
<td>Boot priority for VMs (higher value boots first)</td>
</tr>
<tr>
<td>gvrp</td>
<td>bool</td>
<td>false</td>
<td>no</td>
<td>Register VLAN using GARP VLAN Registration Protocol</td>
</tr>
<tr>
<td>hwaddr</td>
<td>string</td>
<td>randomly assigned</td>
<td>no</td>
<td>The MAC address of the new interface</td>
</tr>
<tr>
<td>maas.subnet.ipv4</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>MAAS IPv4 subnet to register the instance in</td>
</tr>
<tr>
<td>maas.subnet.ipv6</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>MAAS IPv6 subnet to register the instance in</td>
</tr>
<tr>
<td>mtu</td>
<td>integer</td>
<td>parent MTU</td>
<td>no</td>
<td>The MTU of the new interface</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>kernel assigned</td>
<td>no</td>
<td>The name of the interface inside the instance</td>
</tr>
<tr>
<td>network</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>The managed network to link the device to (instead of specifying the nictype directly)</td>
</tr>
<tr>
<td>parent</td>
<td>string</td>
<td>-</td>
<td>yes</td>
<td>The name of the host device (required if specifying the nictype directly)</td>
</tr>
<tr>
<td>vlan</td>
<td>integer</td>
<td>-</td>
<td>no</td>
<td>The VLAN ID to attach to</td>
</tr>
</tbody>
</table>

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:
<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gvrp</td>
<td>bool</td>
<td>false</td>
<td>Register VLAN using GARP VLAN Registration Protocol</td>
</tr>
<tr>
<td>hwaddr</td>
<td>string</td>
<td>randomly assigned</td>
<td>The MAC address of the new interface</td>
</tr>
<tr>
<td>ipv4.address</td>
<td>string</td>
<td>-</td>
<td>Comma-delimited list of IPv4 static addresses to add to the instance (in 12 mode, these can be specified as CIDR values or singular addresses using a subnet of /24)</td>
</tr>
<tr>
<td>ipv4.gateway</td>
<td>string</td>
<td>auto (l3s), (l2)</td>
<td>In l3s mode, whether to add an automatic default IPv4 gateway (can be auto or none); in l2 mode, the IPv4 address of the gateway</td>
</tr>
<tr>
<td>ipv6.host_table</td>
<td>integer</td>
<td>-</td>
<td>The custom policy routing table ID to add IPv6 static routes to (in addition to the main routing table)</td>
</tr>
<tr>
<td>ipv6.address</td>
<td>string</td>
<td>-</td>
<td>Comma-delimited list of IPv6 static addresses to add to the instance (in 12 mode, these can be specified as CIDR values or singular addresses using a subnet of /64)</td>
</tr>
<tr>
<td>ipv6.gateway</td>
<td>string</td>
<td>auto (l3s), (l2)</td>
<td>In l3s mode, whether to add an automatic default IPv6 gateway (can be auto or none); in l2 mode, the IPv6 address of the gateway</td>
</tr>
<tr>
<td>ipv6.host_table</td>
<td>integer</td>
<td>-</td>
<td>The custom policy routing table ID to add IPv6 static routes to (in addition to the main routing table)</td>
</tr>
<tr>
<td>mode</td>
<td>string</td>
<td>13s</td>
<td>The IPVLAN mode (either 12 or 13s)</td>
</tr>
<tr>
<td>mtu</td>
<td>integer</td>
<td>parent MTU</td>
<td>The MTU of the new interface</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>kernel assigned</td>
<td>The name of the interface inside the instance</td>
</tr>
<tr>
<td>parent</td>
<td>string</td>
<td>-</td>
<td>The name of the host device (required)</td>
</tr>
<tr>
<td>vlan</td>
<td>integer</td>
<td>-</td>
<td>The VLAN ID to attach to</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boot.</td>
<td>integer</td>
<td>-</td>
<td>Boot priority for VMs (higher value boots first)</td>
</tr>
<tr>
<td>host.</td>
<td>string</td>
<td>randomly assigned</td>
<td>The name of the interface inside the host</td>
</tr>
<tr>
<td>hwadd</td>
<td>string</td>
<td>randomly assigned</td>
<td>The MAC address of the new interface</td>
</tr>
<tr>
<td>ipv4.</td>
<td>string</td>
<td>-</td>
<td>Comma-delimited list of IPv4 static routes to add on host to NIC</td>
</tr>
<tr>
<td>ipv6.</td>
<td>string</td>
<td>-</td>
<td>Comma-delimited list of IPv6 static routes to add on host to NIC</td>
</tr>
<tr>
<td>limit:</td>
<td>string</td>
<td>-</td>
<td>I/O limit in bit/s for outgoing traffic (various suffixes supported, see Units for storage and network limits)</td>
</tr>
<tr>
<td>egress:</td>
<td>string</td>
<td>-</td>
<td>I/O limit in bit/s for incoming traffic (various suffixes supported, see Units for storage and network limits)</td>
</tr>
<tr>
<td>limit:</td>
<td>string</td>
<td>-</td>
<td>I/O limit in bit/s for both incoming and outgoing traffic (same as setting both limits. ingress and limits.egress)</td>
</tr>
<tr>
<td>max</td>
<td>integer</td>
<td>-</td>
<td>The skb-&gt;priority value (32-bit unsigned integer) for outgoing traffic, to be used by the kernel queuing discipline (qdisc) to prioritize network packets (The effect of this value depends on the particular qdisc implementation, for example, SKBPRI0 or QFQ. Consult the kernel qdisc documentation before setting this value.)</td>
</tr>
<tr>
<td>mtu</td>
<td>integer</td>
<td>kernel assigned</td>
<td>The MTU of the new interface</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>kernel assigned</td>
<td>The name of the interface inside the instance</td>
</tr>
<tr>
<td>queue tx.</td>
<td>integer</td>
<td>-</td>
<td>The transmit queue length for the NIC</td>
</tr>
</tbody>
</table>
**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**).

**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:
Device options

NIC devices of type `routed` have the following device options:

```bash
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
```
<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gvrp</td>
<td>bool</td>
<td>false</td>
<td>Register VLAN using GARP VLAN Registration Protocol</td>
</tr>
<tr>
<td>host_name</td>
<td>string</td>
<td>randomly assigned</td>
<td>The name of the interface inside the host</td>
</tr>
<tr>
<td>hwaddr</td>
<td>string</td>
<td>randomly assigned</td>
<td>The MAC address of the new interface</td>
</tr>
<tr>
<td>ipv4. address</td>
<td>string</td>
<td>-</td>
<td>Comma-delimited list of IPv4 static addresses to add to the instance</td>
</tr>
<tr>
<td>ipv4. gateway</td>
<td>string</td>
<td>auto</td>
<td>Whether to add an automatic default IPv4 gateway (can be auto or none)</td>
</tr>
<tr>
<td>ipv4. host_address</td>
<td>string</td>
<td>169.254.0.1</td>
<td>The IPv4 address to add to the host-side veth interface</td>
</tr>
<tr>
<td>ipv4. host_table</td>
<td>integer</td>
<td>-</td>
<td>The custom policy routing table ID to add IPv4 static routes to (in addition to the main routing table)</td>
</tr>
<tr>
<td>ipv4. neighbor_probe</td>
<td>bool</td>
<td>true</td>
<td>Whether to probe the parent network for IP address availability</td>
</tr>
<tr>
<td>ipv4. routes</td>
<td>string</td>
<td>-</td>
<td>Comma-delimited list of IPv4 static routes to add on host to NIC (without L2 ARP/NDP proxy)</td>
</tr>
<tr>
<td>ipv6. address</td>
<td>string</td>
<td>-</td>
<td>Comma-delimited list of IPv6 static addresses to add to the instance</td>
</tr>
<tr>
<td>ipv6. gateway</td>
<td>string</td>
<td>auto</td>
<td>Whether to add an automatic default IPv6 gateway (can be auto or none)</td>
</tr>
<tr>
<td>ipv6. host_address</td>
<td>string</td>
<td>fe80::1</td>
<td>The IPv6 address to add to the host-side veth interface</td>
</tr>
<tr>
<td>ipv6. host_table</td>
<td>integer</td>
<td>-</td>
<td>The custom policy routing table ID to add IPv6 static routes to (in addition to the main routing table)</td>
</tr>
<tr>
<td>ipv6. neighbor_probe</td>
<td>bool</td>
<td>true</td>
<td>Whether to probe the parent network for IP address availability</td>
</tr>
<tr>
<td>ipv6. routes</td>
<td>string</td>
<td>-</td>
<td>Comma-delimited list of IPv6 static routes to add on host to NIC (without L2 ARP/NDP proxy)</td>
</tr>
<tr>
<td>limits.egress</td>
<td>string</td>
<td>-</td>
<td>I/O limit in bit/s for outgoing traffic (various suffixes supported, see Units for storage and network limits)</td>
</tr>
<tr>
<td>limits.ingress</td>
<td>string</td>
<td>-</td>
<td>I/O limit in bit/s for incoming traffic (various suffixes supported, see Units for storage and network limits)</td>
</tr>
<tr>
<td>limits.max</td>
<td>string</td>
<td>-</td>
<td>I/O limit in bit/s for both incoming and outgoing traffic (same as setting both limits.ingress and limits.egress)</td>
</tr>
<tr>
<td>limits.priority</td>
<td>integer</td>
<td>-</td>
<td>The skb-&gt;priority value (32-bit unsigned integer) for outgoing traffic, to be used by the kernel queueing discipline (qdisc) to prioritize network packets (The effect of this value depends on the particular qdisc implementation, for example, SKBPRI0 or QFQ. Consult the kernel qdisc documentation before setting this value.)</td>
</tr>
<tr>
<td>mtu</td>
<td>string</td>
<td>parent MTU</td>
<td>The MTU of the new interface</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>kernel-assigned</td>
<td>The name of the interface inside the instance</td>
</tr>
<tr>
<td>parent</td>
<td>string</td>
<td>-</td>
<td>The name of the host device to join the instance to</td>
</tr>
<tr>
<td>queue.t.x</td>
<td>string</td>
<td>-</td>
<td>The transmit queue length for the NIC</td>
</tr>
</tbody>
</table>
Canonical LXD

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 and maas.api.key, and then set the 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:

Storage volume

The most common type of disk device is a 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` command to Attach the volume to an instance. Both commands use the same mechanism to add a storage volume as a disk device.

Path on the host
You can share a path on your host (either a file system or a block device) to your instance by adding it as a disk device with the host path as the source:

```
lxc config device add <instance_name> <device_name> disk source=<path_on_host>[
  \[path=<path_in_instance>\]  
]
```

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 with the following command:

```
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=
  \[path_in_instance\]  
]
```

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 with the following command:

```
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
You can add an ISO file as a disk device for a virtual machine. It is added as a ROM device inside the VM. This source type is applicable only to VMs.

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
You can generate a cloud-init configuration ISO from the `cloud-init.vendor-data` and `cloud-init.user-data` configuration keys and attach it to a virtual machine. 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.

To add such a device, use the following command:

```
lxc config device add <instance_name> <device_name> disk source=cloud-init: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` 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.

**Device options**

disk devices have the following device options:
<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Require</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boot.</td>
<td>integer</td>
<td>-</td>
<td>no</td>
<td>Boot priority for VMs (higher value boots first)</td>
</tr>
<tr>
<td>priority</td>
<td>string</td>
<td>cep</td>
<td>no</td>
<td>The cluster name of the Ceph cluster (required for Ceph or CephFS sources)</td>
</tr>
<tr>
<td>ceph.</td>
<td>string</td>
<td>admin</td>
<td>no</td>
<td>The user name of the Ceph cluster (required for Ceph or CephFS sources)</td>
</tr>
<tr>
<td>cluster</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>initial</td>
<td>n/a</td>
<td>-</td>
<td>no</td>
<td>Initial volume configuration for instance root disk devices that allows setting unique configurations independent of default storage pool settings</td>
</tr>
<tr>
<td>io.</td>
<td>string</td>
<td>none</td>
<td>no</td>
<td>Only for VMs: Override the caching mode for the device (none, writeback or unsafe)</td>
</tr>
<tr>
<td>cache</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>limits.</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>I/O limit in byte/s or IOPS for both read and write (same as setting both limits. read and limits.write)</td>
</tr>
<tr>
<td>max</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>limits.</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>I/O limit in byte/s (various suffixes supported, see Units for storage and network limits) or in IOPS (must be suffixed with iops) - see also Configure I/O limits</td>
</tr>
<tr>
<td>read</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>limits.</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>I/O limit in byte/s (various suffixes supported, see Units for storage and network limits) or in IOPS (must be suffixed with iops) - see also Configure I/O limits</td>
</tr>
<tr>
<td>write</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>path</td>
<td>string</td>
<td>-</td>
<td>yes</td>
<td>Path inside the instance where the disk will be mounted (only for containers)</td>
</tr>
<tr>
<td>pool</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>The storage pool to which the disk device belongs (only applicable for storage volumes managed by LXD)</td>
</tr>
<tr>
<td>propagation</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>Controls how a bind-mount is shared between the instance and the host (can be one of private, the default, or shared, unbindable, rshared, rslave, runbindable, rprivate; see the Linux Kernel shared subtree documentation for a full explanation)</td>
</tr>
<tr>
<td>raw.</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>File system specific mount options</td>
</tr>
<tr>
<td>mount.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>options</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>readonly</td>
<td>bool</td>
<td>false</td>
<td>no</td>
<td>Controls whether to make the mount read-only</td>
</tr>
<tr>
<td>recursive</td>
<td>bool</td>
<td>false</td>
<td>no</td>
<td>Controls whether to recursively mount the source path</td>
</tr>
<tr>
<td>require</td>
<td>bool</td>
<td>true</td>
<td>no</td>
<td>Controls whether to fail if the source doesn’t exist</td>
</tr>
<tr>
<td>shift</td>
<td>bool</td>
<td>false</td>
<td>no</td>
<td>Sets up a shifting overlay to translate the source UID/GID to match the instance (only for containers)</td>
</tr>
<tr>
<td>size</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>Disk size in bytes (various suffixes supported, see Units for storage and network limits) - only supported for the rootfs (/)</td>
</tr>
<tr>
<td>size.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>state</td>
<td>string</td>
<td>-</td>
<td>no</td>
<td>Same as size, but applies to the file-system volume used for saving runtime state in VMs</td>
</tr>
<tr>
<td>source</td>
<td>string</td>
<td>-</td>
<td>yes</td>
<td>Source of a file system or block device (see Types of disk devices for details)</td>
</tr>
</tbody>
</table>
**Canonical LXD**

**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 instance (under `/dev`). You can read from the device and write to it.

**Device options**

*unix-char* devices have the following device options:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gid</td>
<td>int</td>
<td>0</td>
<td>GID of the device owner in the instance</td>
</tr>
<tr>
<td>major</td>
<td>int</td>
<td>device on host</td>
<td>Device major number</td>
</tr>
<tr>
<td>minor</td>
<td>int</td>
<td>device on host</td>
<td>Device minor number</td>
</tr>
<tr>
<td>mode</td>
<td>int</td>
<td>0660</td>
<td>Mode of the device in the instance</td>
</tr>
<tr>
<td>path</td>
<td>string</td>
<td>-</td>
<td>Path inside the instance (one of source and path must be set)</td>
</tr>
<tr>
<td>required</td>
<td>bool</td>
<td>true</td>
<td>Whether this device is required to start the instance (see <em>Hotplugging</em>)</td>
</tr>
<tr>
<td>source</td>
<td>string</td>
<td>-</td>
<td>Path on the host (one of source and path must be set)</td>
</tr>
<tr>
<td>uid</td>
<td>int</td>
<td>0</td>
<td>UID of the device owner in the instance</td>
</tr>
</tbody>
</table>

**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 instance (under `/dev`). You can read from the device and write to it.

**Device options**

*unix-block* devices have the following device options:
<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gid</td>
<td>int</td>
<td>0</td>
<td>GID of the device owner in the instance</td>
</tr>
<tr>
<td>major</td>
<td>int</td>
<td>device on host</td>
<td>Device major number</td>
</tr>
<tr>
<td>minor</td>
<td>int</td>
<td>device on host</td>
<td>Device minor number</td>
</tr>
<tr>
<td>mode</td>
<td>int</td>
<td>0660</td>
<td>Mode of the device in the instance</td>
</tr>
<tr>
<td>path</td>
<td>string</td>
<td>-</td>
<td>Path inside the instance (one of source and path must be set)</td>
</tr>
<tr>
<td>required</td>
<td>bool</td>
<td>true</td>
<td>Whether this device is required to start the instance (see Hotplugging)</td>
</tr>
<tr>
<td>source</td>
<td>string</td>
<td>-</td>
<td>Path on the host (one of source and path must be set)</td>
</tr>
<tr>
<td>uid</td>
<td>int</td>
<td>0</td>
<td>UID of the device owner in the instance</td>
</tr>
</tbody>
</table>

**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` or a `unix-hotplug device` 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:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gid</td>
<td>int</td>
<td>0</td>
<td>Only for containers: GID of the device owner in the instance</td>
</tr>
<tr>
<td>mode</td>
<td>int</td>
<td>0660</td>
<td>Only for containers: Mode of the device in the instance</td>
</tr>
<tr>
<td>productid</td>
<td>string</td>
<td>-</td>
<td>The product ID of the USB device</td>
</tr>
<tr>
<td>required</td>
<td>bool</td>
<td>false</td>
<td>Whether this device is required to start the instance (the default is false, and all devices can be hotplugged)</td>
</tr>
<tr>
<td>uid</td>
<td>int</td>
<td>0</td>
<td>Only for containers: UID of the device owner in the instance</td>
</tr>
<tr>
<td>vendorid</td>
<td>string</td>
<td>-</td>
<td>The vendor ID of the USB device</td>
</tr>
</tbody>
</table>
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** (container and VM): Passes an entire GPU through into the instance. This value is the default if gputype is unspecified.
- **mdev** (VM only): Creates and passes a virtual GPU through into the instance.
- **mig** (container only): Creates and passes a MIG (Multi-Instance GPU) through into the instance.
- **sriov** (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:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gid</td>
<td>int</td>
<td>0</td>
<td>GID of the device owner in the instance (container only)</td>
</tr>
<tr>
<td>id</td>
<td>string</td>
<td>-</td>
<td>The DRM card ID of the GPU device</td>
</tr>
<tr>
<td>mode</td>
<td>int</td>
<td>0660</td>
<td>Mode of the device in the instance (container only)</td>
</tr>
<tr>
<td>pci</td>
<td>string</td>
<td>-</td>
<td>The PCI address of the GPU device</td>
</tr>
<tr>
<td>productid</td>
<td>string</td>
<td>-</td>
<td>The product ID of the GPU device</td>
</tr>
<tr>
<td>uid</td>
<td>int</td>
<td>0</td>
<td>UID of the device owner in the instance (container only)</td>
</tr>
<tr>
<td>vendorid</td>
<td>string</td>
<td>-</td>
<td>The vendor ID of the GPU device</td>
</tr>
</tbody>
</table>
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 through into the instance. You can check the list of available mdev profiles by running `lxc info --resources`.

**Device options**

GPU devices of type mdev have the following device options:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>string</td>
<td>-</td>
<td>The DRM card ID of the GPU device</td>
</tr>
<tr>
<td>mdev</td>
<td>string</td>
<td>-</td>
<td>The mdev profile to use (required - for example, i915-GVTg_V5_4)</td>
</tr>
<tr>
<td>pci</td>
<td>string</td>
<td>-</td>
<td>The PCI address of the GPU device</td>
</tr>
<tr>
<td>productid</td>
<td>string</td>
<td>-</td>
<td>The product ID of the GPU device</td>
</tr>
<tr>
<td>vendorid</td>
<td>string</td>
<td>-</td>
<td>The vendor ID of the GPU device</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>string</td>
<td>-</td>
<td>The DRM card ID of the GPU device</td>
</tr>
<tr>
<td>mig.ci</td>
<td>int</td>
<td>-</td>
<td>Existing MIG compute instance ID</td>
</tr>
<tr>
<td>mig.gi</td>
<td>int</td>
<td>-</td>
<td>Existing MIG GPU instance ID</td>
</tr>
<tr>
<td>mig.uuid</td>
<td>string</td>
<td>-</td>
<td>Existing MIG device UUID (MIG- prefix can be omitted)</td>
</tr>
<tr>
<td>pci</td>
<td>string</td>
<td>-</td>
<td>The PCI address of the GPU device</td>
</tr>
<tr>
<td>productid</td>
<td>string</td>
<td>-</td>
<td>The product ID of the GPU device</td>
</tr>
<tr>
<td>vendorid</td>
<td>string</td>
<td>-</td>
<td>The vendor ID of the GPU device</td>
</tr>
</tbody>
</table>

You must set either `mig.uuid` (NVIDIA drivers 470+) or both `mig.ci` and `mig.gi` (old NVIDIA drivers).
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:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>string</td>
<td>-</td>
<td>The DRM card ID of the parent GPU device</td>
</tr>
<tr>
<td>pci</td>
<td>string</td>
<td>-</td>
<td>The PCI address of the parent GPU device</td>
</tr>
<tr>
<td>productid</td>
<td>string</td>
<td>-</td>
<td>The product ID of the parent GPU device</td>
</tr>
<tr>
<td>vendorid</td>
<td>string</td>
<td>-</td>
<td>The vendor ID of the parent GPU device</td>
</tr>
</tbody>
</table>

**Type: infiniband**

Note: The infiniband device type is supported for both containers and VMs. It supports hotplugging 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.

To create a physical infiniband device, use the following command:

```
lxc config device add <instance_name> <device_name> infiniband nictype=physical parent=<device>
```

To create an sriov infiniband device, use the following command:

```
lxc config device add <instance_name> <device_name> infiniband nictype=sriov parent=<sriov_enabled_device>
```
Device options

Infiniband devices have the following device options:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hwaddr</td>
<td>string</td>
<td>randomly assigned</td>
<td>no</td>
<td>The MAC address of the new interface (can be either the full 20-byte variant or the short 8-byte variant, which will only modify the last 8 bytes of the parent device)</td>
</tr>
<tr>
<td>mtu</td>
<td>integer</td>
<td>parent MTU</td>
<td>no</td>
<td>The MTU of the new interface</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>kernel assigned</td>
<td>no</td>
<td>The name of the interface inside the instance</td>
</tr>
<tr>
<td>nicty</td>
<td>string</td>
<td>-</td>
<td>yes</td>
<td>The device type (one of physical or sriov)</td>
</tr>
<tr>
<td>parer</td>
<td>string</td>
<td>-</td>
<td>yes</td>
<td>The name of the host device or bridge</td>
</tr>
</tbody>
</table>

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 forwarding network connections between host and instance. This method makes it possible to forward traffic hitting one of the host’s addresses to an address inside the instance, or to do the reverse and have an address in the instance connect through the host.

In NAT mode, a proxy device can be used for TCP and UDP proxying. In non-NAT mode, you can also proxy traffic between Unix sockets (which can be useful to, for example, forward graphical GUI or audio traffic from the container to the host system) or even across protocols (for example, you can have a TCP listener on the host system and forward its traffic to a Unix socket inside a container).

The supported connection types are:

- tcp <-> tcp
- udp <-> udp
- unix <-> unix
- tcp <-> unix
- unix <-> tcp
- udp <-> tcp
- tcp <-> udp
- udp <-> unix
- 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>
```
NAT mode

The proxy device also supports a NAT mode (`nat=true`), where packets are forwarded using NAT rather than being proxied through a separate connection. This mode has the benefit that the client address is maintained without the need for the target destination to support the HAProxy PROXY protocol (which is the only way to pass the client address through when using the proxy device in non-NAT mode).

However, NAT mode is supported only if the host that the instance is running on is the gateway (which is the case if you’re using `lxdbbr0`, 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 the square bracket notation, for 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 (`0.0.0.0` for IPv4 and `[::]` for IPv6).

**Note:** The listen address can also use wildcard addresses when using 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:
### Key

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bind</td>
<td>string</td>
<td>host</td>
<td>no</td>
<td>Which side to bind on (host/instance)</td>
</tr>
<tr>
<td>connect</td>
<td>string</td>
<td>-</td>
<td>yes</td>
<td>The address and port to connect to</td>
</tr>
<tr>
<td>gid</td>
<td>int</td>
<td>0</td>
<td>no</td>
<td>GID of the owner of the listening Unix socket</td>
</tr>
<tr>
<td>listen</td>
<td>string</td>
<td>-</td>
<td>yes</td>
<td>The address and port to bind and listen</td>
</tr>
<tr>
<td>mode</td>
<td>int</td>
<td>0644</td>
<td>no</td>
<td>Mode for the listening Unix socket</td>
</tr>
<tr>
<td>nat</td>
<td>bool</td>
<td>false</td>
<td>no</td>
<td>Whether to optimize proxying via NAT (requires that the instance NIC has a static IP address)</td>
</tr>
<tr>
<td>proxy_protocol</td>
<td>bool</td>
<td>false</td>
<td>no</td>
<td>Whether to use the HAProxy PROXY protocol to transmit sender information</td>
</tr>
<tr>
<td>security gid</td>
<td>int</td>
<td>0</td>
<td>no</td>
<td>What GID to drop privilege to</td>
</tr>
<tr>
<td>security uid</td>
<td>int</td>
<td>0</td>
<td>no</td>
<td>What UID to drop privilege to</td>
</tr>
<tr>
<td>uid</td>
<td>int</td>
<td>0</td>
<td>no</td>
<td>UID of the owner of the listening Unix socket</td>
</tr>
</tbody>
</table>

### 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 instance (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:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gid</td>
<td>int</td>
<td>0</td>
<td>GID of the device owner in the instance</td>
</tr>
<tr>
<td>mode</td>
<td>int</td>
<td>0660</td>
<td>Mode of the device in the instance</td>
</tr>
<tr>
<td>productid</td>
<td>string</td>
<td>-</td>
<td>The product ID of the Unix device</td>
</tr>
<tr>
<td>required</td>
<td>bool</td>
<td>false</td>
<td>Whether this device is required to start the instance (the default is false, and all devices can be hotplugged)</td>
</tr>
<tr>
<td>uid</td>
<td>int</td>
<td>0</td>
<td>UID of the device owner in the instance</td>
</tr>
<tr>
<td>vendorid</td>
<td>string</td>
<td>-</td>
<td>The vendor ID of the Unix device</td>
</tr>
</tbody>
</table>
**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:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>string</td>
<td>-</td>
<td>for containers</td>
<td>Only for containers: path inside the instance (for example, <code>/dev/tpm0</code>)</td>
</tr>
<tr>
<td>pathrm</td>
<td>string</td>
<td>-</td>
<td>for containers</td>
<td>Only for containers: resource manager path inside the instance (for example, <code>/dev/tpmrm0</code>)</td>
</tr>
</tbody>
</table>

**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` or `nic device`).

**Device options**

`pci` devices have the following device options:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>address</td>
<td>string</td>
<td>-</td>
<td>yes</td>
<td>PCI address of the device</td>
</tr>
</tbody>
</table>
**Instance options**

Instance options are configuration options that are directly related to the instance. See *Configure instance options* for instructions on how to set the instance options.

The key/value configuration is namespaced. The following options are available:

- *Miscellaneous options*
- *Boot-related options*
- *cloud-init configuration*
- *Resource limits*
- *Migration options*
- *NVIDIA and CUDA configuration*
- *Raw instance configuration overrides*
- *Security policies*
- *Snapshot scheduling and configuration*
- *Volatile internal data*

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

<table>
<thead>
<tr>
<th>Key</th>
<th>agent.nic_config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td>Live update</td>
<td>no</td>
</tr>
<tr>
<td>Condition</td>
<td>virtual machine</td>
</tr>
</tbody>
</table>

For containers, the name and MTU of the default network interfaces is used for the instance devices. For virtual machines, set this option to `true` to set the name and MTU of the default network interfaces to be the same as the instance devices.

`cluster.evacuate` What to do when evacuating the instance

<table>
<thead>
<tr>
<th>Key</th>
<th>cluster.evacuate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>auto</td>
</tr>
<tr>
<td>Live update</td>
<td>no</td>
</tr>
</tbody>
</table>

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 and restore cluster members* for more information.

**linux.kernel_modules** Kernel modules to load before starting the instance

<table>
<thead>
<tr>
<th>Key:</th>
<th>linux.kernel_modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Live update:</td>
<td>yes</td>
</tr>
<tr>
<td>Condition:</td>
<td>container</td>
</tr>
</tbody>
</table>

Specify the kernel modules as a comma-separated list.

**linux.sysctl.*** Override for the corresponding `sysctl` setting in the container

<table>
<thead>
<tr>
<th>Key:</th>
<th>linux.sysctl.*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Live update:</td>
<td>no</td>
</tr>
<tr>
<td>Condition:</td>
<td>container</td>
</tr>
</tbody>
</table>

**user.*** Free-form user key/value storage

<table>
<thead>
<tr>
<th>Key:</th>
<th>user.*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Live update:</td>
<td>no</td>
</tr>
</tbody>
</table>

User keys can be used in search.

**environment.*** Environment variables for the instance

<table>
<thead>
<tr>
<th>Key:</th>
<th>environment.*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Live update:</td>
<td>yes (exec)</td>
</tr>
</tbody>
</table>

You can export key/value environment variables to the instance. These are then set for `lxc exec`.
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

<table>
<thead>
<tr>
<th>Key:</th>
<th>boot.autostart</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>bool</td>
</tr>
<tr>
<td>Live update:</td>
<td>no</td>
</tr>
</tbody>
</table>

  If set to false, restore the last state.

- **boot.autostart.delay** Delay after starting the instance

<table>
<thead>
<tr>
<th>Key:</th>
<th>boot.autostart.delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>integer</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td>Live update:</td>
<td>no</td>
</tr>
</tbody>
</table>

  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

<table>
<thead>
<tr>
<th>Key:</th>
<th>boot.autostart.priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>integer</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td>Live update:</td>
<td>no</td>
</tr>
</tbody>
</table>

  The instance with the highest value is started first.

- **boot.host_shutdown_timeout** How long to wait for the instance to shut down

<table>
<thead>
<tr>
<th>Key:</th>
<th>boot.host_shutdown_timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>integer</td>
</tr>
<tr>
<td>Default:</td>
<td>30</td>
</tr>
<tr>
<td>Live update:</td>
<td>yes</td>
</tr>
</tbody>
</table>

  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

<table>
<thead>
<tr>
<th>Key:</th>
<th>boot.stop.priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>integer</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td>Live update:</td>
<td>no</td>
</tr>
</tbody>
</table>

  The instance with the highest value is shut down first.
cloud-init configuration

The following instance options control the cloud-init configuration of the instance:

cloud-init.network-config Network configuration for cloud-init

<table>
<thead>
<tr>
<th>Key</th>
<th>cloud-init.network-config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>DHCP on eth0</td>
</tr>
<tr>
<td>Live update</td>
<td>no</td>
</tr>
<tr>
<td>Condition</td>
<td>If supported by image</td>
</tr>
</tbody>
</table>

The content is used as seed value for cloud-init.

cloud-init.user-data User data for cloud-init

<table>
<thead>
<tr>
<th>Key</th>
<th>cloud-init.user-data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>#cloud-config</td>
</tr>
<tr>
<td>Live update</td>
<td>no</td>
</tr>
<tr>
<td>Condition</td>
<td>If supported by image</td>
</tr>
</tbody>
</table>

The content is used as seed value for cloud-init.

cloud-init.vendor-data Vendor data for cloud-init

<table>
<thead>
<tr>
<th>Key</th>
<th>cloud-init.vendor-data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>#cloud-config</td>
</tr>
<tr>
<td>Live update</td>
<td>no</td>
</tr>
<tr>
<td>Condition</td>
<td>If supported by image</td>
</tr>
</tbody>
</table>

The content is used as seed value for cloud-init.

user.network-config Legacy version of cloud-init.network-config

<table>
<thead>
<tr>
<th>Key</th>
<th>user.network-config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>DHCP on eth0</td>
</tr>
<tr>
<td>Live update</td>
<td>no</td>
</tr>
<tr>
<td>Condition</td>
<td>If supported by image</td>
</tr>
</tbody>
</table>

user.user-data Legacy version of cloud-init.user-data

<table>
<thead>
<tr>
<th>Key</th>
<th>user.user-data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>#cloud-config</td>
</tr>
<tr>
<td>Live update</td>
<td>no</td>
</tr>
<tr>
<td>Condition</td>
<td>If supported by image</td>
</tr>
</tbody>
</table>

user.vendor-data Legacy version of cloud-init.vendor-data
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:

<table>
<thead>
<tr>
<th>Key:</th>
<th>limits.cpu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>1 (VMs)</td>
</tr>
<tr>
<td>Live update:</td>
<td>yes</td>
</tr>
</tbody>
</table>

A number or a specific range of CPUs to expose to the instance.

See [CPU pinning](https://linuxcontainers.org/lxd/) for more information.

<table>
<thead>
<tr>
<th>Key:</th>
<th>limits.cpu.allowance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>100%</td>
</tr>
<tr>
<td>Live update:</td>
<td>yes</td>
</tr>
<tr>
<td>Condition:</td>
<td>container</td>
</tr>
</tbody>
</table>

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)](https://linuxcontainers.org/lxd/) for more information.

<table>
<thead>
<tr>
<th>Key:</th>
<th>limits.cpu.nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Live update:</td>
<td>yes</td>
</tr>
</tbody>
</table>

A comma-separated list of NUMA node IDs or ranges to place the instance CPUs on.

See [Allowance and priority (container only)](https://linuxcontainers.org/lxd/) for more information.

<table>
<thead>
<tr>
<th>Key:</th>
<th>limits.cpu.priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
</tbody>
</table>

CPU scheduling priority compared to other instances.
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)* for more information.

### limits.cpu.priority

Priority of the instance’s CPU

<table>
<thead>
<tr>
<th>Key</th>
<th>limits.cpu.priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>integer</td>
</tr>
<tr>
<td>Default</td>
<td>10 (maximum)</td>
</tr>
<tr>
<td>Live update</td>
<td>yes</td>
</tr>
<tr>
<td>Condition</td>
<td>container</td>
</tr>
</tbody>
</table>

Controls how much priority to give to the instance’s I/O requests when under load. Specify an integer between 0 and 10.

### limits.disk.priority

Priority of the instance’s I/O requests

<table>
<thead>
<tr>
<th>Key</th>
<th>limits.disk.priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>integer</td>
</tr>
<tr>
<td>Default</td>
<td>5 (medium)</td>
</tr>
<tr>
<td>Live update</td>
<td>yes</td>
</tr>
</tbody>
</table>

Fixed value (in bytes) to limit the number of 1 GB huge pages. Various suffixes are supported (see *Units for storage and network limits*).

See *Huge page limits* for more information.

### limits.hugepages.1GB

Limit for the number of 1 GB huge pages

<table>
<thead>
<tr>
<th>Key</th>
<th>limits.hugepages.1GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Live update</td>
<td>yes</td>
</tr>
<tr>
<td>Condition</td>
<td>container</td>
</tr>
</tbody>
</table>

### limits.hugepages.1MB

Limit for the number of 1 MB huge pages

<table>
<thead>
<tr>
<th>Key</th>
<th>limits.hugepages.1MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Live update</td>
<td>yes</td>
</tr>
<tr>
<td>Condition</td>
<td>container</td>
</tr>
</tbody>
</table>

Fixed value (in bytes) to limit the number of 1 MB huge pages. Various suffixes are supported (see *Units for storage and network limits*).

See *Huge page limits* for more information.

### limits.hugepages.2MB

Limit for the number of 2 MB huge pages

<table>
<thead>
<tr>
<th>Key</th>
<th>limits.hugepages.2MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Live update</td>
<td>yes</td>
</tr>
<tr>
<td>Condition</td>
<td>container</td>
</tr>
</tbody>
</table>
Fixed value (in bytes) to limit the number of 2 MB huge pages. Various suffixes are supported (see Units for storage and network limits).

See Huge page limits for more information.

limits.hugepages.64KB Limit for the number of 64 KB huge pages

<table>
<thead>
<tr>
<th>Key:</th>
<th>limits.hugepages.64KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Live update:</td>
<td>yes</td>
</tr>
<tr>
<td>Condition:</td>
<td>container</td>
</tr>
</tbody>
</table>

Fixed value (in bytes) to limit the number of 64 KB huge pages. Various suffixes are supported (see Units for storage and network limits).

See Huge page limits for more information.

limits.memory Usage limit for the host’s memory

<table>
<thead>
<tr>
<th>Key:</th>
<th>limits.memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>1Gib (VMs)</td>
</tr>
<tr>
<td>Live update:</td>
<td>yes</td>
</tr>
</tbody>
</table>

Percentage of the host’s memory or a fixed value in bytes. Various suffixes are supported.

See Units for storage and network limits for details.

limits.memory.enforce Whether the memory limit is hard or soft

<table>
<thead>
<tr>
<th>Key:</th>
<th>limits.memory.enforce</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>hard</td>
</tr>
<tr>
<td>Live update:</td>
<td>yes</td>
</tr>
<tr>
<td>Condition:</td>
<td>container</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Key:</th>
<th>limits.memory.hugepages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>Live update:</td>
<td>no</td>
</tr>
<tr>
<td>Condition:</td>
<td>virtual machine</td>
</tr>
</tbody>
</table>

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
**limits.memory.swap.priority** Prevents the instance from being swapped to disk

<table>
<thead>
<tr>
<th>Key</th>
<th>limits.memory.swap.priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>integer</td>
</tr>
<tr>
<td>Default</td>
<td>10 (maximum)</td>
</tr>
<tr>
<td>Live update</td>
<td>yes</td>
</tr>
<tr>
<td>Condition</td>
<td>container</td>
</tr>
</tbody>
</table>

Specify an integer between 0 and 10. The higher the value, the less likely the instance is to be swapped to disk.

**limits.network.priority** Priority of the instance’s network requests

<table>
<thead>
<tr>
<th>Key</th>
<th>limits.network.priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>integer</td>
</tr>
<tr>
<td>Default</td>
<td>0 (minimum)</td>
</tr>
<tr>
<td>Live update</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Important:** This option is deprecated. Use the per-NIC limits.priority option instead.

Controls how much priority to give to the instance’s network requests when under load.

Specify an integer between 0 and 10.

**limits.processes** Maximum number of processes that can run in the instance

<table>
<thead>
<tr>
<th>Key</th>
<th>limits.processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>integer</td>
</tr>
<tr>
<td>Default</td>
<td>empty</td>
</tr>
<tr>
<td>Live update</td>
<td>yes</td>
</tr>
<tr>
<td>Condition</td>
<td>container</td>
</tr>
</tbody>
</table>

If left empty, no limit is set.

**limits.kernel.*** Kernel resources per instance

<table>
<thead>
<tr>
<th>Key</th>
<th>limits.kernel.*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Live update</td>
<td>no</td>
</tr>
<tr>
<td>Condition</td>
<td>container</td>
</tr>
</tbody>
</table>

You can set kernel limits on an instance, for example, you can limit the number of open files. See *Kernel resource limits* for more information.
CPU limits

You have different options to limit CPU usage:

- Set `limits.cpu` to restrict which CPUs the instance can see and use. See CPU pinning for how to set this option.
- Set `limits.cpu.allowance` 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) 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` with a time limit, you should avoid using `limits.cpu` 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` 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` 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` is set to a single integer, LXD allocates multiple vCPUs and exposes them to the guest as full cores. Those vCPUs are not pinned to specific physical cores on the host. The number of vCPUs can be updated while the VM is running.

When `limits.cpu` is set to a range or comma-separated list of CPU IDs (as provided by `lxc info --resources`), the vCPUs are pinned to those physical 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` 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` 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` to `100%`.

`limits.cpu.nodes` 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` 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` 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. 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**

LXD exposes a generic namedpaced key `limits.kernel.*` that can be used to set resource limits for an instance. 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:
<table>
<thead>
<tr>
<th>Key</th>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>limits.kernel.as</td>
<td>RLIMIT_AS</td>
<td>Maximum size of the process’s virtual memory</td>
</tr>
<tr>
<td>limits.kernel.core</td>
<td>RLIMIT_CORE</td>
<td>Maximum size of the process’s core dump file</td>
</tr>
<tr>
<td>limits.kernel.cpu</td>
<td>RLIMIT_CPU</td>
<td>Limit in seconds on the amount of CPU time the process can consume</td>
</tr>
<tr>
<td>limits.kernel.data</td>
<td>RLIMIT_DATA</td>
<td>Maximum size of the process’s data segment</td>
</tr>
<tr>
<td>limits.fsize</td>
<td>RLIMIT_FSIZE</td>
<td>Maximum size of files the process may create</td>
</tr>
<tr>
<td>limits.locks</td>
<td>RLIMIT_LOCKS</td>
<td>Limit on the number of file locks that this process may establish</td>
</tr>
<tr>
<td>limits.memlock</td>
<td>RLIMIT_MEMLOCK</td>
<td>Limit on the number of bytes of memory that the process may lock in RAM</td>
</tr>
<tr>
<td>limits.nice</td>
<td>RLIMIT_NICE</td>
<td>Maximum value to which the process’s nice value can be raised</td>
</tr>
<tr>
<td>limits.memory</td>
<td>RLIMIT_NOFILE</td>
<td>Maximum number of open files for the process</td>
</tr>
<tr>
<td>limits.nproc</td>
<td>RLIMIT_NPROC</td>
<td>Maximum number of processes that can be created for the user of the calling process</td>
</tr>
<tr>
<td>limits.rtprio</td>
<td>RLIMIT_RTPRIO</td>
<td>Maximum value on the real-time-priority that may be set for this process</td>
</tr>
<tr>
<td>limits.sigpending</td>
<td>RLIMIT_SIGPENDING</td>
<td>Maximum number of signals that may be queued for the user of the calling process</td>
</tr>
</tbody>
</table>

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 instance. 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*:

**migration.incremental.memory** Whether to use incremental memory transfer

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Live update</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>migration.incremental.memory</td>
<td>bool</td>
<td>false</td>
<td>yes</td>
<td>container</td>
</tr>
</tbody>
</table>

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
migration.incremental.memory.goal

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

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

Key: migration.stateful
Type: bool
Default: false
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

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.cudarequired CUDA version

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
The specified version expression is used to set `libnvidia-container NVIDIA_REQUIRE_DRIVER`.

**nvidia.require.driver** Whether to pass the host NVIDIA and CUDA runtime libraries into the instance

<table>
<thead>
<tr>
<th>Key:</th>
<th>nvidia.require.driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Live update:</td>
<td>no</td>
</tr>
<tr>
<td>Condition:</td>
<td>container</td>
</tr>
</tbody>
</table>

**Raw instance configuration overrides**

The following instance options allow direct interaction with the backend features that LXD itself uses: `raw.apparmor`

AppArmor profile entries

<table>
<thead>
<tr>
<th>Key:</th>
<th>raw.apparmor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>blob</td>
</tr>
<tr>
<td>Live update:</td>
<td>yes</td>
</tr>
</tbody>
</table>

The specified entries are appended to the generated profile.

**raw.idmap** Raw idmap configuration

<table>
<thead>
<tr>
<th>Key:</th>
<th>raw.idmap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>blobmap</td>
</tr>
<tr>
<td>Live update:</td>
<td>no</td>
</tr>
<tr>
<td>Condition:</td>
<td>unprivileged container</td>
</tr>
</tbody>
</table>

For example: `both 1000 1000`

**raw.lxc** Raw LXC configuration to be appended to the generated one

<table>
<thead>
<tr>
<th>Key:</th>
<th>raw.lxc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>blob</td>
</tr>
<tr>
<td>Live update:</td>
<td>no</td>
</tr>
<tr>
<td>Condition:</td>
<td>container</td>
</tr>
</tbody>
</table>

**raw.qemu** Raw QEMU configuration to be appended to the generated command line

<table>
<thead>
<tr>
<th>Key:</th>
<th>raw.qemu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>blob</td>
</tr>
<tr>
<td>Live update:</td>
<td>no</td>
</tr>
<tr>
<td>Condition:</td>
<td>virtual machine</td>
</tr>
</tbody>
</table>
raw.qemu.conf Addition/override to the generated qemu.conf file

<table>
<thead>
<tr>
<th>Key</th>
<th>raw.qemu.conf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>blob</td>
</tr>
<tr>
<td>Live update</td>
<td>no</td>
</tr>
<tr>
<td>Condition</td>
<td>virtual machine</td>
</tr>
</tbody>
</table>

See Override QEMU configuration for more information.

raw.seccomp Raw Seccomp configuration

<table>
<thead>
<tr>
<th>Key</th>
<th>raw.seccomp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>blob</td>
</tr>
<tr>
<td>Live update</td>
<td>no</td>
</tr>
<tr>
<td>Condition</td>
<td>container</td>
</tr>
</tbody>
</table>

**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 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:
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 About security policies of the instance: security.agent.metrics

Whether the lxd-agent is queried for state information and metrics

<table>
<thead>
<tr>
<th>Key</th>
<th>security.agent.metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>true</td>
</tr>
<tr>
<td>Live update</td>
<td>no</td>
</tr>
<tr>
<td>Condition</td>
<td>virtual machine</td>
</tr>
</tbody>
</table>

security.csm Whether to use a firmware that supports UEFI-incompatible operating systems
When enabling this option, set `security.secureboot` to `false`.

security.devlxd Whether `/dev/lxd` is present in the instance

See `Communication between instance and host` for more information.

security.devlxd.images Controls the availability of the `/1.0/images` API over devlxd

security.idmap.base The base host ID to use for the allocation

Setting this option overrides auto-detection.

security.idmap.isolated Whether to use a unique idmap for this instance

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
security.nesting Whether to support running LXD (nested) inside the instance

<table>
<thead>
<tr>
<th>Key</th>
<th>security.nesting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td>Live update</td>
<td>yes</td>
</tr>
<tr>
<td>Condition</td>
<td>container</td>
</tr>
</tbody>
</table>

security.privileged Whether to run the instance in privileged mode

<table>
<thead>
<tr>
<th>Key</th>
<th>security.privileged</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td>Live update</td>
<td>no</td>
</tr>
<tr>
<td>Condition</td>
<td>container</td>
</tr>
</tbody>
</table>

security.protection.delete Prevents the instance from being deleted

<table>
<thead>
<tr>
<th>Key</th>
<th>security.protection.delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td>Live update</td>
<td>yes</td>
</tr>
</tbody>
</table>

security.protection.shift Whether to protect the file system from being UID/GID shifted

<table>
<thead>
<tr>
<th>Key</th>
<th>security.protection.shift</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td>Live update</td>
<td>yes</td>
</tr>
<tr>
<td>Condition</td>
<td>container</td>
</tr>
</tbody>
</table>

Set this option to true to prevent the instance’s file system from being UID/GID shifted on startup.

security.secureboot Whether UEFI secure boot is enabled with the default Microsoft keys

<table>
<thead>
<tr>
<th>Key</th>
<th>security.secureboot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>true</td>
</tr>
<tr>
<td>Live update</td>
<td>no</td>
</tr>
<tr>
<td>Condition</td>
<td>virtual machine</td>
</tr>
</tbody>
</table>

When disabling this option, consider enabling security.csm.

security.sev Whether AMD SEV (Secure Encrypted Virtualization) is enabled for this VM

<table>
<thead>
<tr>
<th>Key</th>
<th>security.sev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td>Live update</td>
<td>no</td>
</tr>
<tr>
<td>Condition</td>
<td>virtual machine</td>
</tr>
</tbody>
</table>
security.sev.policy.es Whether AMD SEV-ES (SEV Encrypted State) is enabled for this VM

<table>
<thead>
<tr>
<th>Key</th>
<th>security.sev.policy.es</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td>Live update</td>
<td>no</td>
</tr>
<tr>
<td>Condition</td>
<td>virtual machine</td>
</tr>
</tbody>
</table>

security.sev.session.data The guest owner's base64-encoded session blob

<table>
<thead>
<tr>
<th>Key</th>
<th>security.sev.session.data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>true</td>
</tr>
<tr>
<td>Live update</td>
<td>no</td>
</tr>
<tr>
<td>Condition</td>
<td>virtual machine</td>
</tr>
</tbody>
</table>

security.sev.session.dh The guest owner's base64-encoded Diffie-Hellman key

<table>
<thead>
<tr>
<th>Key</th>
<th>security.sev.session.dh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>true</td>
</tr>
<tr>
<td>Live update</td>
<td>no</td>
</tr>
<tr>
<td>Condition</td>
<td>virtual machine</td>
</tr>
</tbody>
</table>

security.syscalls.allow List of syscalls to allow

<table>
<thead>
<tr>
<th>Key</th>
<th>security.syscalls.allow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Live update</td>
<td>no</td>
</tr>
<tr>
<td>Condition</td>
<td>container</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Key</th>
<th>security.syscalls.deny</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Live update</td>
<td>no</td>
</tr>
<tr>
<td>Condition</td>
<td>container</td>
</tr>
</tbody>
</table>

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)

<table>
<thead>
<tr>
<th>Key</th>
<th>security.syscalls.deny_compat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td>Live update</td>
<td>no</td>
</tr>
<tr>
<td>Condition</td>
<td>container</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Key:</th>
<th>security.syscalls.deny_default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>true</td>
</tr>
<tr>
<td>Live update:</td>
<td>no</td>
</tr>
<tr>
<td>Condition:</td>
<td>container</td>
</tr>
</tbody>
</table>

**security.syscalls.intercept.bpf** Whether to handle the bpf() system call

<table>
<thead>
<tr>
<th>Key:</th>
<th>security.syscalls.intercept.bpf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>Live update:</td>
<td>no</td>
</tr>
<tr>
<td>Condition:</td>
<td>container</td>
</tr>
</tbody>
</table>

**security.syscalls.intercept.bpf.devices** Whether to allow BPF programs

<table>
<thead>
<tr>
<th>Key:</th>
<th>security.syscalls.intercept.bpf.devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>Live update:</td>
<td>no</td>
</tr>
<tr>
<td>Condition:</td>
<td>container</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Key:</th>
<th>security.syscalls.intercept.mknod</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>Live update:</td>
<td>no</td>
</tr>
<tr>
<td>Condition:</td>
<td>container</td>
</tr>
</tbody>
</table>

These system calls allow creation of a limited subset of char/block devices.

**security.syscalls.intercept.mount** Whether to handle the mount system call

<table>
<thead>
<tr>
<th>Key:</th>
<th>security.syscalls.intercept.mount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>Live update:</td>
<td>no</td>
</tr>
<tr>
<td>Condition:</td>
<td>container</td>
</tr>
</tbody>
</table>

**security.syscalls.intercept.mount.allowed** File systems that can be mounted
Specify a comma-separated list of file systems that are safe to mount for processes inside the instance.

**security.syscalls.intercept.mount.allowed** File system that should be redirected to FUSE implementation

Specify the mounts of a given file system that should be redirected to their FUSE implementation (for example, ext4=fuse2fs).

**security.syscalls.intercept.mount.fuse** File system that should be redirected to FUSE implementation

**security.syscalls.intercept.mount.shift** Whether to mount shiftfs on top of file systems handled through mount syscall interception

This system call allows increasing process priority.

**security.syscalls.intercept.setxattr** Whether to handle the setxattr system call

This system call allows setting a limited subset of restricted extended attributes.

**security.syscalls.intercept.sysinfo** Whether to handle the sysinfo system call
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:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>snapshots.expiry</code></td>
<td>string</td>
</tr>
</tbody>
</table>

Specify an expression like `1M 2H 3d 4w 5m 6y`.

**snapshots.pattern** Template for the snapshot name

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>snapshots.pattern</code></td>
<td>string</td>
</tr>
</tbody>
</table>

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](#) for more information.

**snapshots.schedule** Schedule for automatic instance snapshots

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>snapshots.schedule</code></td>
<td>string</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>snapshots.schedule.stopped</code></td>
<td>bool</td>
</tr>
</tbody>
</table>

2.4. Reference
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

The following volatile keys are currently used internally by LXD to store internal data specific to an instance:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>volatile.&lt;name&gt;.apply_quota</td>
<td>string</td>
</tr>
<tr>
<td>volatile.&lt;name&gt;.ceph_rbd</td>
<td>string</td>
</tr>
<tr>
<td>volatile.&lt;name&gt;.host_name</td>
<td>string</td>
</tr>
<tr>
<td>volatile.&lt;name&gt;.hwaddr</td>
<td>string</td>
</tr>
<tr>
<td>volatile.&lt;name&gt;.last_state.created</td>
<td>string</td>
</tr>
</tbody>
</table>

The disk quota is applied the next time the instance starts.

The network device MAC address is used when no `hwaddr` property is set on the device itself.

Possible values are true or false.

Network device original MAC
The original MAC that was used when moving a physical device into an instance.

\texttt{volatile.<name>.last_state.hwaddr} Last used IP addresses

Comma-separated list of the last used IP addresses of the network device.

\texttt{volatile.<name>.last_state.ip_addresses} Network device original MTU

The original MTU that was used when moving a physical device into an instance.

\texttt{volatile.<name>.last_state.mtu} VDPA device name

The VDPA device name used when moving a VDPA device file descriptor into an instance.

\texttt{volatile.<name>.last_state.vdpa.name} SR-IOV virtual function original MAC

The original MAC used when moving a VF into an instance.

\texttt{volatile.<name>.last_state.vf.hwaddr} SR-IOV virtual function ID

The ID used when moving a VF into an instance.

\texttt{volatile.<name>.last_state.vf.id} SR-IOV virtual function original spoof check setting

The original spoof check setting used when moving aVF into an instance.

\texttt{volatile.<name>.last_state.vf.spoofcheck} SR-IOV virtual function original VLAN

2.4. Reference
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

The template with the given name is triggered upon next startup.

volatile.base_image Hash of the base image

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

The cluster member that the instance lived on before evacuation.

volatile.idmap.base The first ID in the instance’s primary idmap range

volatile.idmap.current The idmap currently in use by the instance

volatile.idmap.next The idmap to use the next time the instance starts
volatile.idmap.next
Type: string

volatile.last_state.idmap
Serialized instance UID/GID map
Key: volatile.last_state.idmap
Type: string

volatile.last_state.power
Instance state as of last host shutdown
Key: volatile.last_state.power
Type: string

volatile.uuid
Instance UUID
Key: volatile.uuid
Type: string

The instance UUID is globally unique across all servers and projects.

volatile.uuid.generation
Instance generation UUID
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
Key: volatile.vsock_id
Type: string

Note: Volatile keys cannot be set by the user.

Instance properties

Instance properties are set when the instance is created. They cannot be part of a profile.

The following instance properties are available:

<table>
<thead>
<tr>
<th>Property</th>
<th>Read-only</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>yes</td>
<td>Instance name (see Instance name requirements)</td>
</tr>
<tr>
<td>architecture</td>
<td>no</td>
<td>Instance architecture</td>
</tr>
</tbody>
</table>
Instance name requirements

The instance name can be changed only by renaming the instance with the `lxc rename` 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.

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

Explanation:
  • About instances

2.4.7 Internals

Daemon behavior

This specification covers some of the LXD daemon’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`.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LXD_DIR</td>
<td>The LXD data directory</td>
</tr>
<tr>
<td>LXD_INSECUR</td>
<td>If set to true, allows all default Go ciphers both for client &lt;-&gt; server communication and server &lt;-&gt; image servers (server &lt;-&gt; server and clustering are not affected)</td>
</tr>
<tr>
<td>PATH</td>
<td>List of paths to look into when resolving binaries</td>
</tr>
<tr>
<td>http_proxy</td>
<td>Proxy server URL for HTTP</td>
</tr>
<tr>
<td>https_proxy</td>
<td>Proxy server URL for HTTPS</td>
</tr>
<tr>
<td>no_proxy</td>
<td>List of domains, IP addresses or CIDR ranges that don’t require the use of a proxy</td>
</tr>
</tbody>
</table>

Client environment variable

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDITOR</td>
<td>What text editor to use</td>
</tr>
<tr>
<td>VISUAL</td>
<td>What text editor to use (if EDITOR isn’t set)</td>
</tr>
<tr>
<td>LXD_CONF</td>
<td>Path to the LXC configuration directory</td>
</tr>
<tr>
<td>LXD_GLOBAL_CONF</td>
<td>Path to the global LXC configuration directory</td>
</tr>
<tr>
<td>LXC_REMOTE</td>
<td>Name of the remote to use (overrides configured default remote)</td>
</tr>
</tbody>
</table>
Server environment variable

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LXD_EXEC_PATH</td>
<td>Full path to the LXD binary (used when forking subcommands)</td>
</tr>
<tr>
<td>LXD_LXC_TEMPLATES_PATH</td>
<td>Path to the LXC template configuration directory</td>
</tr>
<tr>
<td>LXD_SECURITY_APPARMOR</td>
<td>If set to false, forces AppArmor off</td>
</tr>
<tr>
<td>LXD_UNPRIVILEGED_ONLY</td>
<td>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.</td>
</tr>
<tr>
<td>LXD_OVMF_PATH</td>
<td>Path to an OVMF build including OVMF_CODE.fd and OVMF_VARS.ms.fd</td>
</tr>
<tr>
<td>LXD_SHIFTFS_DISABLE</td>
<td>Disable shiftfs support (useful when testing traditional UID shifting)</td>
</tr>
<tr>
<td>LXD_IDMAPPI</td>
<td>Disable idmapped mounts support (useful when testing traditional UID shifting)</td>
</tr>
<tr>
<td>LXD_DEVMONITOR_DIR</td>
<td>Path to be monitored by the device monitor. This is primarily for testing.</td>
</tr>
</tbody>
</table>

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` 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` and `security.syscalls.intercept.bpf.devices` 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` 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` 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` 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` 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.
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` to `true`.

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.

### ldmaps 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

On most hosts, LXD will check `/etc/subuid` and `/etc/subgid` for allocations for the `lxd` 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` and `security.idmap.size`.

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`, 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.

Related topics

How-to guides:

- Troubleshooting
2.4.8 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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--all</td>
<td>Show less common commands</td>
</tr>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- **lxc alias** - Manage command aliases
- **lxc cluster** - Manage cluster members
- **lxc config** - Manage instance and server configuration options
- **lxc console** - Attach to instance consoles
- **lxc copy** - Copy instances within or in between LXD servers
- **lxc delete** - Delete instances and snapshots
- **lxc exec** - Execute commands in instances
- **lxc export** - Export instance backups
- **lxc file** - Manage files in instances
- **lxc image** - Manage images
- **lxc import** - Import instance backups
- **lxc info** - Show instance or server information
- **lxc init** - Create instances from images
- **lxc launch** - Create and start instances from images
- **lxc list** - List instances
- **lxc monitor** - Monitor a local or remote LXD server
• **lxc move** - Move instances within or in between LXD servers
• **lxc network** - Manage and attach instances to networks
• **lxc operation** - List, show and delete background operations
• **lxc pause** - Pause instances
• **lxc profile** - Manage profiles
• **lxc project** - Manage projects
• **lxc publish** - Publish instances as images
• **lxc query** - Send a raw query to LXD
• **lxc rebuild** - Rebuild instances
• **lxc remote** - Manage the list of remote servers
• **lxc rename** - Rename instances and snapshots
• **lxc restart** - Restart instances
• **lxc restore** - Restore instances from snapshots
• **lxc snapshot** - Create instance snapshots
• **lxc start** - Start instances
• **lxc stop** - Stop instances
• **lxc storage** - Manage storage pools and volumes
• **lxc version** - Show local and remote versions
• **lxc warning** - Manage warnings

**lxc alias**

Manage command aliases

**Synopsis**

Description: Manage command aliases

```
  lxc alias [flags]
```

**Options inherited from parent commands**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>
SEE ALSO

- `lxc` - Command line client for LXD
- `lxc alias add` - Add new aliases
- `lxc alias list` - List aliases
- `lxc alias remove` - Remove aliases
- `lxc alias rename` - 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` - 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
-h, --help Print help
--project Override the source project
-q, --quiet Don't show progress information
--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 - 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` - 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** - Manage command aliases

**lxc cluster**

Manage cluster members

**Synopsis**

Description: Manage cluster members

```
  lxc cluster [flags]
```

**Options inherited from parent commands**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- **lxc** - Command line client for LXD
- **lxc cluster add** - Request a join token for adding a cluster member
- **lxc cluster edit** - Edit cluster member configurations as YAML
- **lxc cluster enable** - Enable clustering on a single non-clustered LXD server
- **lxc cluster evacuate** - Evacuate cluster member
- **lxc cluster get** - Get values for cluster member configuration keys
- **lxc cluster group** - Manage cluster groups
- **lxc cluster info** - Show useful information about a cluster member
- **lxc cluster list** - List all the cluster members
- **lxc cluster list-tokens** - List all active cluster member join tokens
- **lxc cluster remove** - Remove a member from the cluster
- **lxc cluster rename** - Rename a cluster member
- **lxc cluster restore** - Restore cluster member
- **lxc cluster revoke-token** - Revoke cluster member join token
- **lxc cluster role** - Manage cluster roles
- `lxc cluster set` - Set a cluster member's configuration keys
- `lxc cluster show` - Show details of a cluster member
- `lxc cluster unset` - Unset a cluster member's configuration keys
- `lxc cluster update-certificate` - Update cluster certificate

### 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>:]<name>] [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` - Manage cluster members

### lxc cluster edit

Edit cluster member configurations as YAML
## Synopsis

Description: Edit cluster member configurations as YAML

```
lxc cluster edit [<remote>:]<cluster 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` - 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 the LXD is already available on the network. You can check that by running `lxc config get core.https_address`, and possibly set a value for the address if not yet set.

```
lxc cluster enable [<remote>:] <name> [flags]
```

Chapter 2. Project and community
Options inherited from parent commands

- --debug Show all debug messages
- --force-local Force using the local unix socket
- --help Print help
- --project Override the source project
- --quiet Don't show progress information
- --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 - Manage cluster members

lxc cluster evacuate

Evacuate cluster member

Synopsis

Description: Evacuate cluster member

lxc cluster evacuate [<remote>:]<member> [flags]

Options

- --action Force a particular evacuation action
- --force Force evacuation without user confirmation

Options inherited from parent commands

- --debug Show all debug messages
- --force-local Force using the local unix socket
- --help Print help
- --project Override the source project
- --quiet Don't show progress information
- --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` - 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` - 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 - Manage cluster members
- lxc cluster group add - Add member to group
- lxc cluster group assign - Assign sets of groups to cluster members
- lxc cluster group create - Create a cluster group
- lxc cluster group delete - Delete a cluster group
- lxc cluster group edit - Edit a cluster group
- lxc cluster group list - List all the cluster groups
- lxc cluster group remove - Remove member from group
- lxc cluster group rename - Rename a cluster group
- lxc cluster group show - 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]
Canonical LXD

Options inherited from parent commands

- --debug Show all debug messages
  --force-local Force using the local unix socket
  -h, --help Print help
  --project Override the source project
  -q, --quiet Don't show progress information
  --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 - 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* - Manage cluster groups

**lxc cluster group create**

Create a cluster group

**Synopsis**

Description: Create a cluster group

```
lxc cluster 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
- --quiet Don't show progress information
- --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* - 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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

* lxc cluster group - 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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

* lxc cluster group - 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
-q, --quiet Don't show progress information
--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 - 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
- `--help` Print help
- `--project` Override the source project
- `--quiet` Don't show progress information
- `--sub-commands` Use with help or --help to view sub-commands
- `--version` Show all information messages
- `--version` Print version number

SEE ALSO

- `lxc cluster group` - 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
- `--help` Print help
- `--project` Override the source project
- `--quiet` Don't show progress information
- `--sub-commands` Use with help or --help to view sub-commands
- `--version` Show all information messages
- `--version` Print version number

SEE ALSO

- `lxc cluster group` - 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` - 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 - 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 - 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
-quiet      Don't show progress information
--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` - 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
-`-v, --verbose`  Show all information messages
-`--version`  Print version number

SEE ALSO

- `lxc cluster` - 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` - Manage cluster members
**lxc cluster restore**

Restore cluster member

**Synopsis**

Description: Restore cluster member

```
lxc cluster restore [<remote>:]<member> [flags]
```

**Options**

```
--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` - 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` - 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` - Manage cluster members
- `lxc cluster role add` - Add roles to a cluster member
- `lxc cluster role remove` - 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* - 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

2.4. Reference
SEE ALSO

• lxc cluster role - 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 - 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` - 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
-q, --quiet  Don't show progress information
--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 - 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 - Manage cluster members

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 - Command line client for LXD
- lxc config device - Manage devices
- lxc config edit - Edit instance or server configurations as YAML
- lxc config get - Get values for instance or server configuration keys
- lxc config metadata - Manage instance metadata files
- lxc config set - Set instance or server configuration keys
- lxc config show - Show instance or server configurations
- lxc config template - Manage instance file templates
- lxc config trust - Manage trusted clients
- lxc config unset - Unset instance or server configuration keys

lxc config device

Manage devices

Synopsis

Description: Manage devices

lxc config device [flags]
SEE ALSO

- `lxc config` - Manage instance and server configuration options
- `lxc config device add` - Add instance devices
- `lxc config device get` - Get values for device configuration keys
- `lxc config device list` - List instance devices
- `lxc config device override` - Copy profile inherited devices and override configuration keys
- `lxc config device remove` - Remove instance devices
- `lxc config device set` - Set device configuration keys
- `lxc config device show` - Show full device configuration
- `lxc config device unset` - 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.
```

```
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` - 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
- `--help` Print help
- `--project` Override the source project
- `--quiet` Don't show progress information
- `--sub-commands` Use with help or --help to view sub-commands
- `--verbose` Show all information messages
- `--version` Print version number

SEE ALSO

- `lxc config device` - Manage devices

**lxc config device list**

List instance devices

**Synopsis**

Description: List instance devices

```
lxc config device list [<remote>:]<instance> [flags]
```
Canonical LXD

Options inherited from parent commands

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- lxc config device - 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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- lxc config device - 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 - 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)
-v, --verbose
  Show all information messages
--version
  Print version number

SEE ALSO

- lxc config device - 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 - 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
- `-q`, `--quiet` Don't show progress information
- `--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` - 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
Canonical LXD

Options inherited from parent commands

- --debug Show all debug messages
- --force-local Force using the local unix socket
- -h, --help Print help
- --project Override the source project
- --quiet Don't show progress information
- --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 - 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
- --quiet Don't show progress information
- --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` - 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
-v, --verbose Show all information messages
--version Print version number
```

SEE ALSO

- `lxc config` - Manage instance and server configuration options
- `lxc config metadata edit` - Edit instance metadata files
- `lxc config metadata show` - 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` - 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` - 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.

    lxc config set core.trust_password=blah
    Will set the server's trust password to blah.
```

**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* - 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**

- `-d, --debug` Show all debug messages
- `-f, --force-local` Force using the local unix socket
- `-h, --help` Print help
- `-p, --project` Override the source project
- `-q, --quiet` Don't show progress information
- `-s, --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* - Manage instance and server configuration options

**lxc config template**

Manage instance file templates
Synopsis

Description: Manage instance file templates

```
1xc 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
--project    Override the source project
-q, --quiet  Don't show progress information
--sub-commands Use with help or --help to view sub-commands
-v, --verbose Show all information messages
--version    Print version number
```

SEE ALSO

- `1xc config` - Manage instance and server configuration options
- `1xc config template create` - Create new instance file templates
- `1xc config template delete` - Delete instance file templates
- `1xc config template edit` - Edit instance file templates
- `1xc config template list` - List instance file templates
- `1xc config template show` - Show content of instance file templates

`1xc config template create`

Create new instance file templates

Synopsis

Description: Create new instance file templates

```
1xc config template create [<remote>::]<instance> <template> [flags]
```
Options inherited from parent commands

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- `lxc config template` - 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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- `lxc config template` - 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 - 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 - 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 - 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 - Manage instance and server configuration options
- lxc config trust add - Add new trusted client
- lxc config trust edit - Edit trust configurations as YAML
- lxc config trust list - List trusted clients
- lxc config trust list-tokens - List all active certificate add tokens
- lxc config trust remove - Remove trusted client
- lxc config trust revoke-token - Revoke certificate add token
- lxc config trust show - 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
- `--quiet` Don't show progress information
- `--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` - 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` - 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` - 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` - Manage trusted clients

**lxc config trust remove**

Remove trusted client
Synopsis

Description: Remove trusted client

```
xlc 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
--quiet       Don't show progress information
--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` - Manage trusted clients

```
xlc config trust revoke-token
```

Revoke certificate add token

Synopsis

Description: Revoke certificate add token

```
xlc 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
--quiet       Don't show progress information
--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 - 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 - Manage trusted clients

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

- `--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
- `--help` Print help
- `--project` Override the source project
- `--quiet` Don't show progress information
- `--sub-commands` Use `with` help or `--help` to view sub-commands
- `--verbose` Show all information messages
- `--version` Print version number

SEE ALSO

- `lxc config` - 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 instance's console log
- `--type` Type of connection to establish: 'console' for serial console, 'vga' for SPICE graphical output (default "console")
Canonical LXD

Options inherited from parent commands

- --debug Show all debug messages
- --force-local Force using the local unix socket
- -h, --help Print help
- --project Override the source project
- -q, --quiet Don't show progress information
- --sub-commands Use with help or --help to view sub-commands
- -v, --verbose Show all information messages
- --version Print version number

SEE ALSO

- lxc - 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 - Command line client for LXD

lxc delete

Delete instances and snapshots

Synopsis

Description: Delete instances and snapshots

```bash
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 - 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**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--cwd</td>
<td>Directory to run the command in (default /root)</td>
</tr>
<tr>
<td>-n, --disable-stdin</td>
<td>Disable stdin (reads from /dev/null)</td>
</tr>
<tr>
<td>--env</td>
<td>Environment variable to set (e.g. HOME=/home/foo)</td>
</tr>
<tr>
<td>-t, --force-interactive</td>
<td>Force pseudo-terminal allocation</td>
</tr>
<tr>
<td>-T, --force-noninteractive</td>
<td>Disable pseudo-terminal allocation</td>
</tr>
<tr>
<td>--group</td>
<td>Group ID to run the command as (default 0)</td>
</tr>
<tr>
<td>--mode</td>
<td>Override the terminal mode (auto, interactive or non-interactive) (default &quot;auto&quot;)</td>
</tr>
<tr>
<td>--user</td>
<td>User ID to run the command as (default 0)</td>
</tr>
</tbody>
</table>

**Options inherited from parent commands**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>
SEE ALSO

- lxc - Command line client for LXD

lxc export

Export instance backups

Synopsis

Description: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.

Synopsis: Export instances as backup tarballs.
SEE ALSO

- `lxc` - 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
- `--help` Print help
- `--project` Override the source project
- `--quiet` Don't show progress information
- `--sub-commands` Use `with help or --help` to view sub-commands
- `--verbose` Show all information messages
- `--version` Print version number

SEE ALSO

- `lxc` - Command line client for LXD
- `lxc file delete` - Delete files in instances
- `lxc file edit` - Edit files in instances
- `lxc file mount` - Mount files from instances
- `lxc file pull` - Pull files from instances
- `lxc file push` - Push files into 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` - 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` - 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` - 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` - 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` - 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 - Command line client for LXD
• lxc image alias - Manage image aliases
• lxc image copy - Copy images between servers
• lxc image delete - Delete images
• lxc image edit - Edit image properties
• lxc image export - Export and download images
• lxc image get-property - Get image properties
• lxc image import - Import images into the image store
• lxc image info - Show useful information about images
• lxc image list - List images
• lxc image refresh - Refresh images
• lxc image set-property - Set image properties
• lxc image show - Show image properties

2.4. Reference 327
lxc image unset-property
- 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 - Manage images
- lxc image alias create - Create aliases for existing images
- lxc image alias delete - Delete image aliases
- lxc image alias list - List image aliases
- lxc image alias rename - 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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use <code>with help</code> or <code>--help</code> to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- `lxc image alias` - 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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use <code>with help</code> or <code>--help</code> to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- `lxc image alias` - 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` - 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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- lxc image alias - 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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--alias</td>
<td>New aliases to add to the image</td>
</tr>
<tr>
<td>--auto-update</td>
<td>Keep the image up to date after initial copy</td>
</tr>
<tr>
<td>--copy-aliases</td>
<td>Copy aliases from source</td>
</tr>
<tr>
<td>--mode</td>
<td>Transfer mode. One of pull (default), push or relay (default</td>
</tr>
<tr>
<td>-p, --profile</td>
<td>Profile to apply to the new image</td>
</tr>
<tr>
<td>--public</td>
<td>Make image public</td>
</tr>
<tr>
<td>--target-project</td>
<td>Copy to a project different from the source</td>
</tr>
<tr>
<td>--vm</td>
<td>Copy virtual machine images</td>
</tr>
</tbody>
</table>
Canonical LXD

Options inherited from parent commands

```
--debug Show all debug messages
--force-local Force using the local unix socket
-h, --help Print help
--project Override the source project
-q, --quiet Don't show progress information
--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` - 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` - 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 - 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` - 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
```
SEE ALSO

- `lxc image` - 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.

```
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` - 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` - 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
- a - Architecture
- s - Size
- u - Upload date
- t - Type

```
lxc image list [<remote>:] [<filter>...] [flags]
```

Options

```
-c, --columns Columns (default "lfpdasu")
-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 - Manage images

lxc image refresh

Refresh images

Synopsis

Description: Refresh images

```
lxc image refresh [<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

SEE ALSO

- lxc image - 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 - 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
- q, --quiet Don't show progress information
- --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 - 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 - 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

```bash
lxc import backup0.tar.gz
Create a new instance using backup0.tar.gz as the source.
```

Options

```
-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` - 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.
```
Canonical LXD

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
-h, --help   Print help
--project    Override the source project
-q, --quiet  Don't show progress information
--sub-commands Use with help or --help to view sub-commands
-v, --verbose Show all information messages
--version    Print version number
```

SEE ALSO

- `lxc` - 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:22.04 u1

lxc init ubuntu:22.04 u1 < config.yaml
    Create the instance with configuration from config.yaml
```
Options

- `--config` Config key/value to apply to the new instance
- `--device` New key/value to apply to a specific device
- `--empty` Create an empty instance
- `--ephemeral` Ephemeral instance
- `--network` Network name
- `--no-profiles` Create the instance with no profiles applied
- `--profile` Profile to apply to the new instance
- `--storage` Storage pool name
- `--target` Cluster member name
- `--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
- `--help` Print help
- `--project` Override the source project
- `--quiet` Don't show progress information
- `--sub-commands` Use with help or --help to view sub-commands
- `--verbose` Show all information messages
- `--version` Print version number

SEE ALSO

- `lxc` - 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:22.04 u1
lxc launch ubuntu:22.04 u1 < config.yaml
Create and start a container with configuration from config.yaml

lxc launch ubuntu:22.04 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:22.04 v1 --vm -c limits.cpu=4 -c limits.memory=4GiB
Create and start a virtual machine with 4 vCPUs and 4GiB of RAM
```

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** - 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 by “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 (persistent or ephemeral) 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` - 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 - 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 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` - 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` - Command line client for LXD
- `lxc network acl` - Manage network ACLs
- `lxc network attach` - Attach network interfaces to instances
- `lxc network attach-profile` - Attach network interfaces to profiles
- `lxc network create` - Create new networks
- `lxc network delete` - Delete networks

2.4. Reference
- **lxc network detach** - Detach network interfaces from instances
- **lxc network detach-profile** - Detach network interfaces from profiles
- **lxc network edit** - Edit network configurations as YAML
- **lxc network forward** - Manage network forwards
- **lxc network get** - Get values for network configuration keys
- **lxc network info** - Get runtime information on networks
- **lxc network list** - List available networks
- **lxc network list-allocations** - List network allocations in use
- **lxc network list-leases** - List DHCP leases
- **lxc network load-balancer** - Manage network load balancers
- **lxc network peer** - Manage network peerings
- **lxc network rename** - Rename networks
- **lxc network set** - Set network configuration keys
- **lxc network show** - Show network configurations
- **lxc network unset** - Unset network configuration keys
- **lxc network zone** - Manage network zones

### lxc network acl

Manage network ACLs

#### Synopsis

Description: Manage network ACLs

```plaintext
lxc network acl [flags]
```

#### Options inherited from parent commands

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>
SEE ALSO

- `lxc network` - Manage and attach instances to networks
- `lxc network acl create` - Create new network ACLs
- `lxc network acl delete` - Delete network ACLs
- `lxc network acl edit` - Edit network ACL configurations as YAML
- `lxc network acl get` - Get values for network ACL configuration keys
- `lxc network acl list` - List available network ACLs
- `lxc network acl rename` - Rename network ACLs
- `lxc network acl rule` - Manage network ACL rules
- `lxc network acl set` - Set network ACL configuration keys
- `lxc network acl show` - Show network ACL configurations
- `lxc network acl show-log` - Show network ACL log
- `lxc network acl unset` - 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]
```

**Options inherited from parent commands**

```
--debug    Show all debug messages
--force-local Force using the local unix socket
-h, --help  Print help
--project  Override the source project
-q, --quiet Don't show progress information
--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 - 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
--quiet Don't show progress information
--sub-commands Use with help or --help to view sub-commands
--verbose Show all information messages
--version Print version number
```

SEE ALSO

• lxc network acl - 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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- `lxc network acl` - 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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>
SEE ALSO

- `lxc network acl` - Manage network ACLs

**lxc network acl list**

List available network ACLs

**Synopsis**

Description: List available network ACL

```
lxc network acl 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 network acl` - 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` - 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
```
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 - Manage network ACLs
• lxc network acl rule add - Add rules to an ACL
• lxc network acl rule remove - Remove rules from an ACL
```

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 - 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` - 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
- `--help` Print help
- `--project` Override the source project
- `--quiet` Don't show progress information
- `--sub-commands` Use with help or --help to view sub-commands
- `--verbose` Show all information messages
- `--version` Print version number

SEE ALSO

- `lxc network acl` - 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` - 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` - 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
--project        Override the source project
-q, --quiet      Don't show progress information
--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` - 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
--quiet       Don't show progress information
--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` - 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` - 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` - 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` - 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` - 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` - 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` - Manage and attach instances to networks
- `lxc network forward create` - Create new network forwards
- `lxc network forward delete` - Delete network forwards
- `lxc network forward edit` - Edit network forward configurations as YAML
- `lxc network forward get` - Get values for network forward configuration keys
- `lxc network forward list` - List available network forwards
- `lxc network forward port` - Manage network forward ports
- `lxc network forward set` - Set network forward keys
- `lxc network forward show` - Show network forward configurations
- `lxc network forward unset` - 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]
```

**Options**

```
--target    Cluster member name
```
Canonical LXD

Options inherited from parent commands

- --debug Show all debug messages
- --force-local Force using the local unix socket
- -h, --help Print help
- --project Override the source project
- -q, --quiet Don't show progress information
- --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 - 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 - 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 - Manage network forwards

lxc network forward get

Get values for network forward configuration keys
Canonical LXD

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
--quiet        Don't show progress information
--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` - 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
- --quiet Don't show progress information
- --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 - 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
- --quiet Don't show progress information
- --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 - Manage network forwards
- lxc network forward port add - Add ports to a forward
- lxc network forward port remove - 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` - 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` - 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` - 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 - 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 - 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` - 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` - Manage and attach instances to networks

**lxc network list**

List available networks

**Synopsis**

Description: List available networks

```
lxc network 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 network* - 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* - 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` - 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` - Manage and attach instances to networks
- `lxc network load-balancer backend` - Manage network load balancer backends
- `lxc network load-balancer create` - Create new network load balancers
- `lxc network load-balancer delete` - Delete network load balancers
- `lxc network load-balancer edit` - Edit network load balancer configurations as YAML
- `lxc network load-balancer get` - Get values for network load balancer configuration keys
- `lxc network load-balancer list` - List available network load balancers
- `lxc network load-balancer port` - Manage network load balancer ports
- `lxc network load-balancer set` - Set network load balancer keys
- `lxc network load-balancer show` - Show network load balancer configurations
- `lxc network load-balancer unset` - 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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- `lxc network load-balancer` - Manage network load balancers
- `lxc network load-balancer backend add` - Add backends to a load balancer
- `lxc network load-balancer backend remove` - 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 - 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]
```
Canonical LXD

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` - 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]
```

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` - 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` - 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

```bash
lxc network load-balancer edit [<remote>:]<network> <listen_address> [flags]
```

**Options**

```bash
--target Cluster member name
```

**Options inherited from parent commands**

```bash
--debug Show all debug messages
--force-local Force using the local unix socket
-h, --help Print help
--project Override the source project
-q, --quiet Don't show progress information
--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` - 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

```bash
lxc network load-balancer get [<remote>:]<network> <listen_address> <key> [flags]
```

**Options**

```bash
-p, --property Get the key as a network load balancer property
```
Options inherited from parent commands

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- lxc network load-balancer - 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>;<project>] [flags]
```

Options

- -f, --format Format (csv|json|table|yaml|compact) (default "table")

Options inherited from parent commands

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>
SEE ALSO

- `lxc network load-balancer` - 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**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--debug</code></td>
<td>Show all debug messages</td>
</tr>
<tr>
<td><code>--force-local</code></td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td><code>-h, --help</code></td>
<td>Print help</td>
</tr>
<tr>
<td><code>--project</code></td>
<td>Override the source project</td>
</tr>
<tr>
<td><code>-q, --quiet</code></td>
<td>Don't show progress information</td>
</tr>
<tr>
<td><code>--sub-commands</code></td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td><code>-v, --verbose</code></td>
<td>Show all information messages</td>
</tr>
<tr>
<td><code>--version</code></td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- `lxc network load-balancer` - Manage network load balancers
- `lxc network load-balancer port add` - Add ports to a load balancer
- `lxc network load-balancer port remove` - 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

```bash
lxc network load-balancer port add [<remote>:]<network> <listen_address> <protocol> <listen_port(s)> <backend_name>[,<backend_name>...] [flags]
```
Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--target</td>
<td>Cluster member name</td>
</tr>
</tbody>
</table>

Options inherited from parent commands

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- `lxc network load-balancer port` - 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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--force</td>
<td>Remove all ports that match</td>
</tr>
<tr>
<td>--target</td>
<td>Cluster member name</td>
</tr>
</tbody>
</table>

Options inherited from parent commands

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>
SEE ALSO

- `lxc network load-balancer port` - 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**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-p</code>, <code>--property</code></td>
<td>Set the key as a network load balancer property</td>
</tr>
<tr>
<td><code>--target</code></td>
<td>Cluster member name</td>
</tr>
</tbody>
</table>

**Options inherited from parent commands**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--debug</code></td>
<td>Show all debug messages</td>
</tr>
<tr>
<td><code>--force-local</code></td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td><code>-h</code>, <code>--help</code></td>
<td>Print help</td>
</tr>
<tr>
<td><code>--project</code></td>
<td>Override the source project</td>
</tr>
<tr>
<td><code>-q</code>, <code>--quiet</code></td>
<td>Don't show progress information</td>
</tr>
<tr>
<td><code>--sub-commands</code></td>
<td>Use with <code>help</code> or <code>--help</code> to view sub-commands</td>
</tr>
<tr>
<td><code>-v</code>, <code>--verbose</code></td>
<td>Show all information messages</td>
</tr>
<tr>
<td><code>--version</code></td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- `lxc network load-balancer` - Manage network load balancers

**lxc network load-balancer show**

Show network load balancer configurations
Synopsis

Description: Show network load balancer configurations

```
1xc 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` - 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 - 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 - Manage and attach instances to networks
- lxc network peer create - Create new network peering
- lxc network peer delete - Delete network peerings
- lxc network peer edit - Edit network peer configurations as YAML
- lxc network peer get - Get values for network peer configuration keys
- lxc network peer list - List available network peers
• `lxc network peer set` - Set network peer keys
• `lxc network peer show` - Show network peer configurations
• `lxc network peer unset` - 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` - 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 - 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 - 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
-q, --quiet Don't show progress information
--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 - Manage network peerings

lxc network peer list

List available network peers

Synopsis

Description: List available network peers

lxc network peer list [remote:]network [flags]
Canonical LXD

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 - 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 - 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 - 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

- \( p, \ --\text{property} \) Unset the key as a network peer \text{property}

Options inherited from parent commands

- \( \text{--debug} \) Show all debug messages
- \( \text{--force-local} \) Force using the local unix socket
- \( \text{-h, --help} \) Print help
- \( \text{-project} \) Override the source project
- \( \text{-q, --quiet} \) Don't show progress information
- \( \text{--sub-commands} \) Use with help or --help to view sub-commands
- \( \text{-v, --verbose} \) Show all information messages
- \( \text{--version} \) Print version number

SEE ALSO

- \( \text{lxc network peer} \) - Manage network peerings

lxc network rename

Rename networks

Synopsis

Description: Rename networks

\text{lxc network rename [\text{<remote>:\text{]<network}> \text{<new-name> [flags]}]

Options inherited from parent commands

- \( \text{--debug} \) Show all debug messages
- \( \text{--force-local} \) Force using the local unix socket
- \( \text{-h, --help} \) Print help
- \( \text{-project} \) Override the source project
- \( \text{-q, --quiet} \) Don't show progress information
- \( \text{--sub-commands} \) Use with help or --help to view sub-commands
- \( \text{-v, --verbose} \) Show all information messages
- \( \text{--version} \) Print version number
SEE ALSO

- `lxc network` - 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
- `--help` Print help
- `--project` Override the source project
- `--quiet` Don't show progress information
- `--sub-commands` Use `with help` or `--help` to view sub-commands
- `--verbose` Show all information messages
- `--version` Print version number

SEE ALSO

- `lxc network` - Manage and attach instances to networks

**lxc network show**

Show network configurations
**Canonical LXD**

**Synopsis**

Description: Show network configurations

```
lxc network show [<remote>:]<network> [flags]
```

**Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--target</td>
<td>Cluster member name</td>
</tr>
</tbody>
</table>

**Options inherited from parent commands**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

**SEE ALSO**

- `lxc network` - 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**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-p, --property</td>
<td>Unset the key as a network property</td>
</tr>
<tr>
<td>--target</td>
<td>Cluster member name</td>
</tr>
</tbody>
</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 - 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 - Manage and attach instances to networks
- lxc network zone create - Create new network zones
- lxc network zone delete - Delete network zones
- lxc network zone edit - Edit network zone configurations as YAML
- lxc network zone get - Get values for network zone configuration keys
- lxc network zone list - List available network zoneS
lxc network zone record - Manage network zone records
lxc network zone set - Set network zone configuration keys
lxc network zone show - Show network zone configurations
lxc network zone unset - 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]
```

Options inherited from parent commands

```
--debug          Show all debug messages
--force-local    Force using the local unix socket
-h, --help       Print help
--project        Override the source project
-q, --quiet      Don't show progress information
--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 - 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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use <code>with</code> help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- `lxc network zone` - 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 [remoteREDENTIALS]ZONE FLAGS]
```

Options inherited from parent commands

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use <code>with</code> help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- `lxc network zone` - 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
- `-q, --quiet` Don't show progress information
- `--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` - Manage network zones

lxc network zone list

List available network zones

Synopsis

Description: List available network zones

lxc network zone list [remote::] [flags]
Options

- `--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
- `--help` Print help
- `--project` Override the source project
- `--quiet` Don't show progress information
- `--sub-commands` Use `help` or `--help` to view sub-commands
- `--verbose` Show all information messages
- `--version` Print version number

SEE ALSO

- `lxc network zone` - 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
- `--help` Print help
- `--project` Override the source project
- `--quiet` Don't show progress information
- `--sub-commands` Use `help` or `--help` to view sub-commands
- `--verbose` Show all information messages
- `--version` Print version number
SEE ALSO

- `lxc network zone` - Manage network zones
- `lxc network zone record create` - Create new network zone record
- `lxc network zone record delete` - Delete network zone record
- `lxc network zone record edit` - Edit network zone record configurations as YAML
- `lxc network zone record entry` - Manage network zone record entries
- `lxc network zone record get` - Get values for network zone record configuration keys
- `lxc network zone record list` - List available network zone records
- `lxc network zone record set` - Set network zone record configuration keys
- `lxc network zone record show` - Show network zone record configuration
- `lxc network zone record unset` - 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]
```

Options inherited from parent commands

```
--debug
--force-local
-h, --help
--project
-q, --quiet
--sub-commands
-v, --verbose
--version
```

SEE ALSO

- `lxc network zone record` - 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 - 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` - 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**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- `lxc network zone record` - Manage network zone records
- `lxc network zone record entry add` - Add a network zone record entry
- `lxc network zone record entry remove` - 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
--sub-commands  Use with help or --help to view sub-commands
-q, --quiet  Don't show progress information
-v, --verbose  Show all information messages
--version  Print version number

SEE ALSO

- lxc network zone record entry - 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
--sub-commands  Use with help or --help to view sub-commands
-q, --quiet  Don't show progress information
-v, --verbose  Show all information messages
--version  Print version number
SEE ALSO

- `lxc network zone record entry` - 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` - 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 - 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 - 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 - 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** - 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 - 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` - 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` - Manage network zones

**lxc operation**

List, show and delete background operations
Canonical LXD

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 - Command line client for LXD
- lxc operation delete - Delete a background operation (will attempt to cancel)
- lxc operation list - List background operations
- lxc operation show - 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` - 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` - List, show and delete background operations

### lxc operation show

Show details on a background operation
Canonical LXD

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` - 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` - 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` - Command line client for LXD
  - `lxc profile add` - Add profiles to instances
  - `lxc profile assign` - Assign sets of profiles to instances
  - `lxc profile copy` - Copy profiles
  - `lxc profile create` - Create profiles
  - `lxc profile delete` - Delete profiles
**Canonical LXD**

- `lxc profile device` - Manage devices
- `lxc profile edit` - Edit profile configurations as YAML
- `lxc profile get` - Get values for profile configuration keys
- `lxc profile list` - List profiles
- `lxc profile remove` - Remove profiles from instances
- `lxc profile rename` - Rename profiles
- `lxc profile set` - Set profile configuration keys
- `lxc profile show` - Show profile configurations
- `lxc profile unset` - 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**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use <strong>with help</strong> or <strong>--help</strong> to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

**SEE ALSO**

- `lxc profile` - 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
--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` - 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` - Manage profiles

```
$ lxc profile create [<remote>[:]<profile>] [flags]
```

Create profiles

Synopsis

Description: Create profiles

Options inherited from parent commands

```
--debug Show all debug messages
--force-local Force using the local unix socket
-h, --help Print 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)
SEE ALSO

- `lxc profile` - Manage profiles

**lxc profile delete**

Delete profiles

**Synopsis**

Description: Delete profiles

```
```

**Options inherited from parent commands**

```

**SEE ALSO**

- `lxc profile` - 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` - Manage profiles
- `lxc profile device add` - Add instance devices
- `lxc profile device get` - Get values for device configuration keys
- `lxc profile device list` - List instance devices
- `lxc profile device remove` - Remove instance devices
- `lxc profile device set` - Set device configuration keys
- `lxc profile device show` - Show full device configuration
- `lxc profile device unset` - 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` - 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 - 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 - 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 - 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 - 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
-v, --verbose Show all information messages
--version Print version number

SEE ALSO

• lxc profile device - 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* - 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
- `--quiet` Don't show progress information
- `--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* - Manage profiles

**lxc profile get**

Get values for profile configuration keys
Canonical LXD

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` - Manage profiles

### lxc profile list

List profiles

Synopsis

Description: List profiles

```
$ lxc profile 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 profile` - 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
-q, --quiet Don't show progress information
--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` - 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 - 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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use <code>with</code> help or <code>--help</code> to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

### SEE ALSO

- `lxc profile` - Manage profiles

### lxc profile show

Show profile configurations

#### Synopsis

Description: Show profile configurations

```
$ lxc profile show [<remote>:]<profile> [flags]
```

#### Options inherited from parent commands

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use <code>with</code> help or <code>--help</code> to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

### SEE ALSO

- `lxc profile` - Manage profiles
Canonical LXD

**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 - 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** - Command line client for LXD
- **lxc project create** - Create projects
- **lxc project delete** - Delete projects
- **lxc project edit** - Edit project configurations as YAML
- **lxc project get** - Get values for project configuration keys
- **lxc project info** - Get a summary of resource allocations
- **lxc project list** - List projects
- **lxc project rename** - Rename projects
- **lxc project set** - Set project configuration keys
- **lxc project show** - Show project options
- **lxc project switch** - Switch the current project
- **lxc project unset** - Unset project configuration keys

**lxc project create**

Create projects

**Synopsis**

Description: Create projects

```
lxc project create [<remote>:]<project> [flags]
```
Canonical LXD

Options

-c, --config Config key/value to apply to the new project

Options inherited from parent commands

- --debug Show all debug messages
  --force-local Force using the local unix socket
  -h, --help Print help
  --project Override the source project
  -q, --quiet Don't show progress information
  --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 - 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` - 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` - Manage projects

**lxc project get**

Get values for project configuration keys
Canonical LXD

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` - Manage projects

**lxc project info**

Get a summary of resource allocations

Synopsis

Description: Get a summary of resource allocations

```
lxc project info [<remote>:]<project> <key> [flags]
```

Options

- `f, --format`  Format (csv|json|table|yaml|compact) (default "table")
Options inherited from parent commands

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- `lxc project` - Manage projects

lxc project list

List projects

Synopsis

Description: List projects

```
lxc project list [<remote>::] [flags]
```

Options

- `--format` Format (csv|json|table|yaml|compact) (default "table")

Options inherited from parent commands

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>
SEE ALSO

- `lxc project` - 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` - 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

- `--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
- `--help` Print help
- `--project` Override the source project
- `--quiet` Don't show progress information
- `--sub-commands` Use `with help or --help to view sub-commands`
- `--verbose` Show all information messages
- `--version` Print version number

SEE ALSO

- `lxc project` - Manage projects

`lxc project show`

Show project options

Synopsis

Description: Show project options

`lxc project show [<remote>:]<project> [flags]`
**SEE ALSO**

- *lxc project* - 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* - 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
- `--help` Print help
- `--project` Override the source project
- `--sub-commands` Use with `help` or `--help` to view sub-commands
- `--quiet` Don't show progress information
- `--version` Print version number

SEE ALSO

- `lxc project` - 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)
- `--force` Stop the instance if currently running
- `--public` Make the image public
- `--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 - 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 (defaults to GET) (default "GET")
- --wait Wait for the operation to complete
Options inherited from parent commands

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- lxc - 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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>
SEE ALSO

- lxc - 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 - Command line client for LXD
- lxc remote add - Add new remote servers
- lxc remote get-default - Show the default remote
- lxc remote list - List the available remotes
- lxc remote remove - Remove remotes
- lxc remote rename - Rename remotes
- lxc remote set-url - Set the URL for the remote
- lxc remote switch - 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**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--accept-certificate</td>
<td>Accept certificate</td>
</tr>
<tr>
<td>--auth-type</td>
<td>Server authentication type (tls, candid, or oidc)</td>
</tr>
<tr>
<td>--domain</td>
<td>Candid domain to use</td>
</tr>
<tr>
<td>--password</td>
<td>Remote admin password</td>
</tr>
<tr>
<td>--project</td>
<td>Project to use for the remote</td>
</tr>
<tr>
<td>--protocol</td>
<td>Server protocol (lxd or simplestreams)</td>
</tr>
<tr>
<td>--public</td>
<td>Public image server</td>
</tr>
</tbody>
</table>

**Options inherited from parent commands**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

**SEE ALSO**

- *lxc remote* - 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` - 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 - 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 - Manage the list of remote servers
Canonical LXD

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 - 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` - 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` - 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]
```
Canonical LXD

Options inherited from parent commands

```
--debug Show all debug messages
--force-local Force using the local unix socket
-h, --help Print help
--project Override the source project
-q, --quiet Don't show progress information
--sub-commands Use with help or --help to view sub-commands
-v, --verbose Show all information messages
--version Print version number
```

SEE ALSO

• lxc - 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 - 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 - 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, ...

```
[remote]:<instance> [snapshot name] [flags]
```

Examples

```
lxc snapshot u1 snap0
Create a snapshot of "u1" called "snap0".
```

Options

```
--no-expiry
--reuse
--stateful
```

Ignore any configured auto-expiry for the instance
If the snapshot name already exists, delete and create a new one
Whether or not to snapshot the instance’s running state

Options inherited from parent commands

```
--debug
--force-local
--help
--project
--quiet
--sub-commands
--version
```

Show all debug messages
Force using the local unix socket
Print help
Override the source project
Don't show progress information
Use with help or --help to view sub-commands
Show all information messages
Print version number
SEE ALSO

- lxc - 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 - Command line client for LXD

lxc stop

Stop instances
Canonical LXD

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 - 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 - Command line client for LXD
- lxc storage bucket - Manage storage buckets
- lxc storage create - Create storage pools
- lxc storage delete - Delete storage pools
- lxc storage edit - Edit storage pool configurations as YAML
- lxc storage get - Get values for storage pool configuration keys
- lxc storage info - Show useful information about storage pools
- lxc storage list - List available storage pools
- lxc storage set - Set storage pool configuration keys
- lxc storage show - Show storage pool configurations and resources
- lxc storage unset - Unset storage pool configuration keys
- lxc storage volume - Manage storage volumes

lxc storage bucket

Manage storage buckets

Synopsis

Description: Manage storage buckets.

lxc storage bucket [flags]
Options inherited from parent commands

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- `lxc storage` - Manage storage pools and volumes
- `lxc storage bucket create` - Create new custom storage buckets
- `lxc storage bucket delete` - Delete storage buckets
- `lxc storage bucket edit` - Edit storage bucket configurations as YAML
- `lxc storage bucket get` - Get values for storage bucket configuration keys
- `lxc storage bucket key` - Manage storage bucket keys
- `lxc storage bucket list` - List storage buckets
- `lxc storage bucket set` - Set storage bucket configuration keys
- `lxc storage bucket show` - Show storage bucket configurations
- `lxc storage bucket unset` - 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]
```

**Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--target</td>
<td>Cluster member name</td>
</tr>
</tbody>
</table>
Options inherited from parent commands

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- `lxc storage bucket` - 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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>
SEE ALSO

- lxc storage bucket - 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 - 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 - Manage storage buckets

lxc storage bucket key

Manage storage bucket keys

Synopsis

Description: Manage storage bucket keys.

lxc storage bucket key [flags]
Canonical LXD

Options inherited from parent commands

```
--debug           Show all debug messages
--force-local     Force using the local unix socket
-h, --help        Print help
--project         Override the source project
-q, --quiet       Don't show progress information
--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` - Manage storage buckets
- `lxc storage bucket key create` - Create key for a storage bucket
- `lxc storage bucket key delete` - Delete key from a storage bucket
- `lxc storage bucket key edit` - Edit storage bucket key as YAML
- `lxc storage bucket key list` - List storage bucket keys
- `lxc storage bucket key show` - 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]
```

**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` - 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 - 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 key 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**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- lxc storage bucket key - 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 - 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` - Manage storage bucket keys

**lxc storage bucket list**

List storage buckets

**Synopsis**

Description: List storage buckets

```
lxc storage bucket list [<remote>:]<pool> [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 bucket - 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` - 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
-v, --verbose Show all information messages
--version     Print version number
```

SEE ALSO

- `lxc storage bucket` - 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` - Manage storage buckets

lxc storage create

Create storage pools

**Synopsis**

Description: Create storage pools

```
lxc storage create [remote:]pool <driver> [key=value...] [flags]
```
Canonical LXD

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` - 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` - 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` - Manage storage pools and volumes

**lxc storage get**

Get values for storage pool configuration keys
Canonical LXD

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` - 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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>--quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- **lxc storage** - 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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>--quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>
SEE ALSO

- `lxc storage` - 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` - 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**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--resources</td>
<td>Show the resources available to the storage pool</td>
</tr>
<tr>
<td>--target</td>
<td>Cluster member name</td>
</tr>
</tbody>
</table>

**Options inherited from parent commands**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

**SEE ALSO**

- `lxc storage` - 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**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-p, --property</td>
<td>Unset the key as a storage property</td>
</tr>
<tr>
<td>--target</td>
<td>Cluster member name</td>
</tr>
</tbody>
</table>
Canonical LXD

Options inherited from parent commands

```
--debug Show all debug messages
--force-local Force using the local unix socket
-h, --help Print help
--project Override the source project
-q, --quiet Don't show progress information
--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` - 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
-q, --quiet Don't show progress information
--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` - Manage storage pools and volumes
- `lxc storage volume attach` - Attach new storage volumes to instances
- `lxc storage volume attach-profile` - Attach new storage volumes to profiles
- `lxc storage volume copy` - Copy storage volumes
- `lxc storage volume create` - Create new custom storage volumes
- `lxc storage volume delete` - Delete storage volumes
- `lxc storage volume detach` - Detach storage volumes from instances
- `lxc storage volume detach-profile` - Detach storage volumes from profiles
- `lxc storage volume edit` - Edit storage volume configurations as YAML
- `lxc storage volume export` - Export custom storage volume
- `lxc storage volume get` - Get values for storage volume configuration keys
- `lxc storage volume import` - Import custom storage volumes
- `lxc storage volume info` - Show storage volume state information
- `lxc storage volume list` - List storage volumes
- `lxc storage volume move` - Move storage volumes between pools
- `lxc storage volume rename` - Rename storage volumes and storage volume snapshots
- `lxc storage volume restore` - Restore storage volume snapshots
- `lxc storage volume set` - Set storage volume configuration keys
- `lxc storage volume show` - Show storage volume configurations
- `lxc storage volume snapshot` - Snapshot storage volumes
- `lxc storage volume unset` - Unset storage volume configuration keys

### lxc storage volume attach

Attach new storage volumes to instances

#### Synopsis

Description: Attach new storage volumes to instances

```
[lxc storage volume attach [<remote>:]<pool> <volume> <instance> [<device name>] [<path>...]
```

#### Options inherited from parent commands

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>
SEE ALSO

- `lxc storage volume` - Manage storage volumes

### lxc storage volume attach-profile

Attach new storage volumes to profiles

**Synopsis**

Description: Attach new storage volumes to profiles

```
lxc storage volume attach-profile [remote:]<pool> <volume> <profile> [<device name>] [<path>] [flags]
```

**Options inherited from parent commands**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- `lxc storage volume` - 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` - 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]
```

Options

```
--target Cluster member name
--type Content type, block or filesystem (default "filesystem")
```
## Options inherited from parent commands

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

## SEE ALSO

- `lxc storage volume` - 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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>
SEE ALSO

- lxc storage volume - 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> <volume> <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
- --quiet Don't show progress information
- --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 - 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> <volume> <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 - 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* - 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
- --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` - 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 - 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 - 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 - 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: 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 "etndcuL")
-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 volume - Manage storage volumes

lxc storage volume move

Move storage volumes between pools
Canonical LXD

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` - 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 - 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
-v, --verbose Show all information messages
--version Print version number
```
SEE ALSO

- `lxc storage volume` - 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 [:] /

```bash
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.

```bash
lxc storage volume set default data size=1GiB
Sets the size of a custom volume "data" in pool "default" to 1 GiB.
```

```bash
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**

- `--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
- `--help` Print help
- `--project` Override the source project
- `--quiet` Don't show progress information
- `--sub-commands` Use with help or --help to view sub-commands
- `--verbose` Show all information messages
- `--version` Print version number
SEE ALSO

- `lxc storage volume` - 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` - Manage storage volumes

**lxc storage volume snapshot**

Snapshot storage volumes

**Synopsis**

Description: Snapshot storage volumes

```
 lxc storage volume snapshot [<remote>:]<pool> <volume> [<snapshot>] [flags]
```

**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` - 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
  Remotes 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 - Manage storage volumes
```
lxc version

Show local and remote versions

Synopsis

Description: Show local and remote versions

```bash
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` - Command line client for LXD

lxc warning

Manage warnings

Synopsis

Description: Manage warnings

```bash
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` - Command line client for LXD
- `lxc warning acknowledge` - Acknowledge warning
- `lxc warning delete` - Delete warning
- `lxc warning list` - List warnings
- `lxc warning show` - 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` - 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
-v, --verbose   Show all information messages
--version   Print version number
```

SEE ALSO

```
• lxc warning - 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
```
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

- `-d, --debug` Show all debug messages
- `-l, --force-local` Force using the local unix socket
- `-h, --help` Print help
- `-p, --project` Override the source project
- `-q, --quiet` Don't show progress information
- `-s, --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` - Manage warnings

lxc warning show

Show warning

Synopsis

Description: Show warning

lxc warning show [<remote>::<warning-uuid>] [flags]
Options inherited from parent commands

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug</td>
<td>Show all debug messages</td>
</tr>
<tr>
<td>--force-local</td>
<td>Force using the local unix socket</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Print help</td>
</tr>
<tr>
<td>--project</td>
<td>Override the source project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>Don't show progress information</td>
</tr>
<tr>
<td>--sub-commands</td>
<td>Use with help or --help to view sub-commands</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Show all information messages</td>
</tr>
<tr>
<td>--version</td>
<td>Print version number</td>
</tr>
</tbody>
</table>

SEE ALSO

- lxc warning - Manage warnings

2.4.9 Networks

LXD supports different network types for 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

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 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 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:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Condition</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bgp.peers.NAME.address</td>
<td>string</td>
<td>BGP server</td>
<td>-</td>
</tr>
<tr>
<td>bgp.peers.NAME.asn</td>
<td>integer</td>
<td>BGP server</td>
<td>-</td>
</tr>
<tr>
<td>bgp.peers.NAME.password</td>
<td>string</td>
<td>BGP server</td>
<td>- (no password)</td>
</tr>
<tr>
<td>bgp.peers.NAME.holdtime</td>
<td>integer</td>
<td>BGP server</td>
<td>180</td>
</tr>
<tr>
<td>bgp.ipv4.nexthop</td>
<td>string</td>
<td>BGP server</td>
<td>local address</td>
</tr>
<tr>
<td>bgp.ipv6.nexthop</td>
<td>string</td>
<td>BGP server</td>
<td>local address</td>
</tr>
<tr>
<td>bridge.driver</td>
<td>string</td>
<td>-</td>
<td>native</td>
</tr>
<tr>
<td>bridge.external_interfaces</td>
<td>string</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>bridge.hwaddr</td>
<td>string</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>bridge.mode</td>
<td>string</td>
<td>-</td>
<td>standard</td>
</tr>
<tr>
<td>bridge.mtu</td>
<td>integer</td>
<td>-</td>
<td>1500</td>
</tr>
<tr>
<td>dns.domain</td>
<td>string</td>
<td>-</td>
<td>lxd</td>
</tr>
<tr>
<td>Key</td>
<td>Type</td>
<td>Condition</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------</td>
<td>-------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>dns.mode</td>
<td>string</td>
<td>-</td>
<td>managed</td>
</tr>
<tr>
<td>dns.search</td>
<td>string</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>dns.zone.forward</td>
<td>string</td>
<td>-</td>
<td>managed</td>
</tr>
<tr>
<td>dns.zone.reverse.ipv4</td>
<td>string</td>
<td>-</td>
<td>managed</td>
</tr>
<tr>
<td>dns.zone.reverse.ipv6</td>
<td>string</td>
<td>-</td>
<td>managed</td>
</tr>
<tr>
<td>fan.overlay_subnet</td>
<td>string</td>
<td>fan mode</td>
<td>240.0.0.0/8</td>
</tr>
<tr>
<td>fan.type</td>
<td>string</td>
<td>fan mode</td>
<td>vxlan</td>
</tr>
<tr>
<td>fan.underlay_subnet</td>
<td>string</td>
<td>fan mode</td>
<td>auto (on create only)</td>
</tr>
<tr>
<td>ipv4.address</td>
<td>string</td>
<td>standard mode</td>
<td>- (initial value on creation: auto)</td>
</tr>
<tr>
<td>ipv4.dhcp</td>
<td>bool</td>
<td>IPv4 address</td>
<td>true</td>
</tr>
<tr>
<td>ipv4.dhcp.expiry</td>
<td>string</td>
<td>IPv4 DHCP</td>
<td>1h</td>
</tr>
<tr>
<td>ipv4.dhcp.gateway</td>
<td>string</td>
<td>IPv4 DHCP</td>
<td>IPv4 address</td>
</tr>
<tr>
<td>ipv4.dhcp.ranges</td>
<td>string</td>
<td>IPv4 DHCP</td>
<td>all addresses</td>
</tr>
<tr>
<td>ipv4.firewall</td>
<td>bool</td>
<td>IPv4 address</td>
<td>true</td>
</tr>
<tr>
<td>ipv4.nat</td>
<td>bool</td>
<td>IPv4 address</td>
<td>false (initial value on creation if ipv4.address is set to auto)</td>
</tr>
<tr>
<td>ipv4.nat.address</td>
<td>string</td>
<td>IPv4 address</td>
<td>-</td>
</tr>
<tr>
<td>ipv4.nat.order</td>
<td>string</td>
<td>IPv4 address</td>
<td>before</td>
</tr>
<tr>
<td>ipv4.ovn.ranges</td>
<td>string</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ipv4.routes</td>
<td>string</td>
<td>IPv4 address</td>
<td>-</td>
</tr>
<tr>
<td>ipv4.routing</td>
<td>bool</td>
<td>IPv4 address</td>
<td>true</td>
</tr>
<tr>
<td>ipv6.address</td>
<td>string</td>
<td>standard mode</td>
<td>- (initial value on creation: auto)</td>
</tr>
<tr>
<td>ipv6.dhcp</td>
<td>bool</td>
<td>IPv6 address</td>
<td>true</td>
</tr>
<tr>
<td>ipv6.dhcp.expiry</td>
<td>string</td>
<td>IPv6 DHCP</td>
<td>1h</td>
</tr>
<tr>
<td>ipv6.dhcp.ranges</td>
<td>string</td>
<td>IPv6 stateful DHCP</td>
<td>all addresses</td>
</tr>
<tr>
<td>ipv6.firewall</td>
<td>bool</td>
<td>IPv6 address</td>
<td>true</td>
</tr>
<tr>
<td>ipv6.nat</td>
<td>bool</td>
<td>IPv6 address</td>
<td>false (initial value on creation if ipv6.address is set to auto)</td>
</tr>
<tr>
<td>ipv6.nat.address</td>
<td>string</td>
<td>IPv6 address</td>
<td>-</td>
</tr>
<tr>
<td>ipv6.nat.order</td>
<td>string</td>
<td>IPv6 address</td>
<td>before</td>
</tr>
<tr>
<td>ipv6.ovn.ranges</td>
<td>string</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ipv6routes</td>
<td>string</td>
<td>IPv6 address</td>
<td>-</td>
</tr>
<tr>
<td>maas.subnet.ipv4</td>
<td>string</td>
<td>IPv4 address</td>
<td>-</td>
</tr>
<tr>
<td>maas.subnet.ipv6</td>
<td>string</td>
<td>IPv6 address</td>
<td>-</td>
</tr>
<tr>
<td>raw.dnsmasq</td>
<td>string</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>security.acls</td>
<td>string</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>security.acls.default.egress.action</td>
<td>string</td>
<td>security.acls</td>
<td>reject</td>
</tr>
<tr>
<td>security.acls.default.egress.logged</td>
<td>bool</td>
<td>security.acls</td>
<td>false</td>
</tr>
<tr>
<td>security.acls.default.ingress.action</td>
<td>string</td>
<td>security.acls</td>
<td>reject</td>
</tr>
<tr>
<td>security.acls.default.ingress.logged</td>
<td>bool</td>
<td>security.acls</td>
<td>false</td>
</tr>
<tr>
<td>tunnel.NAME.group</td>
<td>string</td>
<td>vxlan</td>
<td>239.0.0.1</td>
</tr>
<tr>
<td>tunnel.NAME.id</td>
<td>integer</td>
<td>vxlan</td>
<td>0</td>
</tr>
<tr>
<td>tunnel.NAME.interface</td>
<td>string</td>
<td>vxlan</td>
<td>-</td>
</tr>
<tr>
<td>tunnel.NAME.local</td>
<td>string</td>
<td>gre or vxlan</td>
<td>-</td>
</tr>
<tr>
<td>tunnel.NAME.port</td>
<td>integer</td>
<td>vxlan</td>
<td>0</td>
</tr>
<tr>
<td>tunnel.NAME.protocol</td>
<td>string</td>
<td>standard mode</td>
<td>-</td>
</tr>
<tr>
<td>tunnel.NAME.remote</td>
<td>string</td>
<td>gre or vxlan</td>
<td>-</td>
</tr>
<tr>
<td>tunnel.NAME.ttl</td>
<td>integer</td>
<td>vxlan</td>
<td>1</td>
</tr>
<tr>
<td>user.*</td>
<td>string</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Supported features

The following features are supported for the bridge network type:

- How to configure network ACLs
- How to configure network forwards
- How to configure network zones
- How to configure LXD as a BGP server
- How to integrate with systemd-resolved

Firewall issues

See How to configure your firewall 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 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 or Physical network 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 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.

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 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:
<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Condition</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>network</td>
<td>string</td>
<td>-</td>
<td>-</td>
<td>Uplink network to use for external network access</td>
</tr>
<tr>
<td>bridge.hwaddr</td>
<td>string</td>
<td>-</td>
<td>-</td>
<td>MAC address for the bridge</td>
</tr>
<tr>
<td>bridge.mtu</td>
<td>integer</td>
<td>-</td>
<td>1442</td>
<td>Bridge MTU (default allows host to host Geneve tunnels)</td>
</tr>
<tr>
<td>dns.domain</td>
<td>string</td>
<td>-</td>
<td>lxd</td>
<td>Domain to advertise to DHCP clients and use for DNS resolution</td>
</tr>
<tr>
<td>dns.search</td>
<td>string</td>
<td>-</td>
<td>-</td>
<td>Full comma-separated domain search list, defaulting to dns.domain value</td>
</tr>
<tr>
<td>dns.zone.forward</td>
<td>string</td>
<td>-</td>
<td>-</td>
<td>Comma-separated list of DNS zone names for forward DNS records</td>
</tr>
<tr>
<td>dns.zone.reverse.ipv4</td>
<td>string</td>
<td>-</td>
<td>-</td>
<td>DNS zone name for IPv4 reverse DNS records</td>
</tr>
<tr>
<td>dns.zone.reverse.ipv6</td>
<td>string</td>
<td>-</td>
<td>-</td>
<td>DNS zone name for IPv6 reverse DNS records</td>
</tr>
<tr>
<td>ipv4.address</td>
<td>string</td>
<td>standard mode</td>
<td>- (initial value on creation: auto)</td>
<td>IPv4 address for the bridge (use none to turn off IPv4 or auto to generate a new random unused subnet) (CIDR)</td>
</tr>
<tr>
<td>ipv4.dhcp</td>
<td>bool</td>
<td>IPv4 address</td>
<td>true</td>
<td>Whether to allocate addresses using DHCP</td>
</tr>
<tr>
<td>ipv4.l3only</td>
<td>bool</td>
<td>IPv4 address</td>
<td>false</td>
<td>Whether to enable layer 3 only mode.</td>
</tr>
<tr>
<td>ipv4.nat</td>
<td>bool</td>
<td>IPv4 address</td>
<td>false (initial value on creation if ipv4.address is set to auto: true)</td>
<td>Whether to NAT</td>
</tr>
<tr>
<td>ipv4.nat.address</td>
<td>string</td>
<td>IPv4 address</td>
<td>-</td>
<td>The source address used for outbound traffic from the network (requires uplink ovn. ingress_mode=routed)</td>
</tr>
<tr>
<td>ipv6.address</td>
<td>string</td>
<td>standard mode</td>
<td>- (initial value on creation: auto)</td>
<td>IPv6 address for the bridge (use none to turn off IPv6 or auto to generate a new random unused subnet) (CIDR)</td>
</tr>
<tr>
<td>ipv6.dhcp</td>
<td>bool</td>
<td>IPv6 address</td>
<td>true</td>
<td>Whether to provide additional network configuration over DHCP</td>
</tr>
<tr>
<td>ipv6.dhcp.stateful</td>
<td>bool</td>
<td>IPv6 DHCP</td>
<td>false</td>
<td>Whether to allocate addresses using DHCP</td>
</tr>
<tr>
<td>ipv6.l3only</td>
<td>bool</td>
<td>IPv6 DHCP</td>
<td>false</td>
<td>Whether to enable layer 3 only mode.</td>
</tr>
<tr>
<td>ipv6.nat</td>
<td>bool</td>
<td>IPv6 address</td>
<td>false (initial value on creation if ipv6.address is set to auto: true)</td>
<td>Whether to NAT</td>
</tr>
<tr>
<td>ipv6.nat.address</td>
<td>string</td>
<td>IPv6 address</td>
<td>-</td>
<td>The source address used for outbound traffic from the network (requires uplink ovn. ingress_mode=routed)</td>
</tr>
<tr>
<td>security.acls</td>
<td>string</td>
<td>-</td>
<td>-</td>
<td>Comma-separated list of Network ACLs to apply to NICs connected to this network</td>
</tr>
<tr>
<td>security.acls.default.egress.action</td>
<td>string</td>
<td>securit reject acls</td>
<td></td>
<td>Action to use for egress traffic that doesn’t match any ACL rule</td>
</tr>
<tr>
<td>security.acls.default.egress.action</td>
<td>bool</td>
<td>securit false acls</td>
<td></td>
<td>Whether to log egress traffic that doesn’t match any ACL rule</td>
</tr>
<tr>
<td>security.acls.default.ingress.action</td>
<td>string</td>
<td>securit reject acls</td>
<td></td>
<td>Action to use for ingress traffic that doesn’t match any ACL rule</td>
</tr>
<tr>
<td>security.acls.default.ingress.action</td>
<td>bool</td>
<td>securit false acls</td>
<td></td>
<td>Whether to log ingress traffic that doesn’t match any ACL rule</td>
</tr>
</tbody>
</table>
Supported features

The following features are supported for the ovn network type:

- How to configure network ACLs
- How to configure network forwards
- How to configure network zones
- How to create OVN peer routing relationships
- How to configure network load balancers

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 sub-interfaces 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 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:
<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Condition</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gvrp</td>
<td>bool</td>
<td>-</td>
<td>false</td>
<td>Register VLAN using GARP VLAN Registration Protocol</td>
</tr>
<tr>
<td>mtu</td>
<td>integer</td>
<td>-</td>
<td>-</td>
<td>The MTU of the new interface</td>
</tr>
<tr>
<td>parent</td>
<td>string</td>
<td>-</td>
<td>-</td>
<td>Parent interface to create macvlan NICs on</td>
</tr>
<tr>
<td>vlan</td>
<td>integer</td>
<td>-</td>
<td>-</td>
<td>The VLAN ID to attach to</td>
</tr>
<tr>
<td>maas.subnet.ipv4</td>
<td>string</td>
<td>IPv4 address</td>
<td>-</td>
<td>MAAS IPv4 subnet to register instances in (when using network property on NIC)</td>
</tr>
<tr>
<td>maas.subnet.ipv6</td>
<td>string</td>
<td>IPv6 address</td>
<td>-</td>
<td>MAAS IPv6 subnet to register instances in (when using network property on NIC)</td>
</tr>
<tr>
<td>user.*</td>
<td>string</td>
<td>-</td>
<td>-</td>
<td>User-provided free-form key/value pairs</td>
</tr>
</tbody>
</table>

### 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 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:
<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Condition</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gvrp</td>
<td>bool</td>
<td>-</td>
<td>false</td>
<td>Register VLAN using GARP VLAN Registration Protocol</td>
</tr>
<tr>
<td>mtu</td>
<td>integer</td>
<td>-</td>
<td>-</td>
<td>The MTU of the new interface</td>
</tr>
<tr>
<td>parent</td>
<td>string</td>
<td>-</td>
<td>-</td>
<td>Existing interface to use for network</td>
</tr>
<tr>
<td>vlan</td>
<td>integer</td>
<td>-</td>
<td>-</td>
<td>The VLAN ID to attach to</td>
</tr>
<tr>
<td>bgp.peers. NAME. address</td>
<td>string</td>
<td>BGP server</td>
<td>-</td>
<td>Peer address (IPv4 or IPv6) for use by ovn downstream networks</td>
</tr>
<tr>
<td>bgp.peers. NAME.asn</td>
<td>integer</td>
<td>BGP server</td>
<td>-</td>
<td>Peer AS number for use by ovn downstream networks</td>
</tr>
<tr>
<td>bgp.peers. NAME. password</td>
<td>string</td>
<td>BGP server</td>
<td>-</td>
<td>Peer session password (optional) for use by ovn downstream networks</td>
</tr>
<tr>
<td>bgp.peers. NAME. holdtime</td>
<td>integer</td>
<td>BGP server</td>
<td>180</td>
<td>Peer session hold time (in seconds; optional)</td>
</tr>
<tr>
<td>dns.nameservers</td>
<td>string</td>
<td>standard mode</td>
<td>-</td>
<td>List of DNS server IPs on physical network</td>
</tr>
<tr>
<td>ipv4.gateway</td>
<td>string</td>
<td>standard mode</td>
<td>-</td>
<td>IPv4 address for the gateway and network (CIDIR)</td>
</tr>
<tr>
<td>ipv4.ovn.ranges</td>
<td>string</td>
<td>-</td>
<td>-</td>
<td>Comma-separated list of IPv4 ranges to use for child OVN network routers</td>
</tr>
<tr>
<td>ipv4.routes</td>
<td>string</td>
<td>IPv4 address</td>
<td>-</td>
<td>Comma-separated list of additional IPv4 CIDR subnets that can be used with child OVN networks ipv4.routes.external setting</td>
</tr>
<tr>
<td>ipv6.gateway</td>
<td>string</td>
<td>standard mode</td>
<td>-</td>
<td>IPv6 address for the gateway and network (CIDIR)</td>
</tr>
<tr>
<td>ipv6.ovn.ranges</td>
<td>string</td>
<td>-</td>
<td>-</td>
<td>Comma-separated list of IPv6 ranges to use for child OVN network routers</td>
</tr>
<tr>
<td>ipv6.routes</td>
<td>string</td>
<td>IPv6 address</td>
<td>-</td>
<td>Comma-separated list of additional IPv6 CIDR subnets that can be used with child OVN networks ipv6.routes.external setting</td>
</tr>
<tr>
<td>ipv6.routes.anycast</td>
<td>bool</td>
<td>IPv6 address</td>
<td>false</td>
<td>Allow the overlapping routes to be used on multiple networks/NIC at the same time</td>
</tr>
<tr>
<td>maas.subnet.ipv4</td>
<td>string</td>
<td>IPv4 address</td>
<td>-</td>
<td>MAAS IPv4 subnet to register instances in (when using network property on NIC)</td>
</tr>
<tr>
<td>maas.subnet.ipv6</td>
<td>string</td>
<td>IPv6 address</td>
<td>-</td>
<td>MAAS IPv6 subnet to register instances in (when using network property on NIC)</td>
</tr>
<tr>
<td>ovn.ingress_mode</td>
<td>string</td>
<td>standard mode</td>
<td>12proxy</td>
<td>Sets the method how OVN NIC external IPs will be advertised on uplink network: 12proxy (proxy ARP/NDP) or routed</td>
</tr>
<tr>
<td>user.*</td>
<td>string</td>
<td>-</td>
<td>-</td>
<td>User-provided free-form key/value pairs</td>
</tr>
</tbody>
</table>
Supported features

The following features are supported for the physical network type:

- *How to configure LXD as a BGP server*

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 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:

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Condition</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mtu</td>
<td>integer</td>
<td>-</td>
<td>-</td>
<td>The MTU of the new interface</td>
</tr>
<tr>
<td>parent</td>
<td>string</td>
<td>-</td>
<td>-</td>
<td>Parent interface to create sriov NICs on</td>
</tr>
<tr>
<td>vlan</td>
<td>integer</td>
<td>-</td>
<td>-</td>
<td>The VLAN ID to attach to</td>
</tr>
<tr>
<td>maas.subnet.ipv4</td>
<td>string</td>
<td>IPv4 address</td>
<td>-</td>
<td>MAAS IPv4 subnet to register instances in (when using network property on NIC)</td>
</tr>
<tr>
<td>maas.subnet.ipv6</td>
<td>string</td>
<td>IPv6 address</td>
<td>-</td>
<td>MAAS IPv6 subnet to register instances in (when using network property on NIC)</td>
</tr>
<tr>
<td>user.*</td>
<td>string</td>
<td>-</td>
<td>-</td>
<td>User-provided free-form key/value pairs</td>
</tr>
</tbody>
</table>
2.4.10 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.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Type</th>
<th>Item</th>
<th>Value</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>soft</td>
<td>nofile</td>
<td>1048576</td>
<td>unset</td>
<td>Maximum number of open files</td>
</tr>
<tr>
<td>*</td>
<td>hard</td>
<td>nofile</td>
<td>1048576</td>
<td>unset</td>
<td>Maximum number of open files</td>
</tr>
<tr>
<td>root</td>
<td>soft</td>
<td>nofile</td>
<td>1048576</td>
<td>unset</td>
<td>Maximum number of open files</td>
</tr>
<tr>
<td>root</td>
<td>hard</td>
<td>nofile</td>
<td>1048576</td>
<td>unset</td>
<td>Maximum number of open files</td>
</tr>
<tr>
<td>*</td>
<td>soft</td>
<td>memlocl</td>
<td>unlimited</td>
<td>unset</td>
<td>Maximum locked-in-memory address space (KB)</td>
</tr>
<tr>
<td>*</td>
<td>hard</td>
<td>memlocl</td>
<td>unlimited</td>
<td>unset</td>
<td>Maximum locked-in-memory address space (KB)</td>
</tr>
<tr>
<td>root</td>
<td>soft</td>
<td>memlocl</td>
<td>unlimited</td>
<td>unset</td>
<td>Maximum locked-in-memory address space (KB), only need with bpf syscall supervision</td>
</tr>
<tr>
<td>root</td>
<td>hard</td>
<td>memlocl</td>
<td>unlimited</td>
<td>unset</td>
<td>Maximum locked-in-memory address space (KB), only need with bpf syscall supervision</td>
</tr>
</tbody>
</table>

/etc/sysctl.conf

Note: Reboot the server after changing any of these parameters.
### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fs.aio-max-nr</td>
<td>5242</td>
<td>6553</td>
<td>Maximum number of concurrent asynchronous I/O operations (you might need to increase this limit further if you have a lot of workloads that use the AIO subsystem, for example, MySQL)</td>
</tr>
<tr>
<td>fs.inotify.max_queued_events</td>
<td>1048</td>
<td>1638</td>
<td>Upper limit on the number of events that can be queued to the corresponding inotify instance (see inotify)</td>
</tr>
<tr>
<td>fs.inotify.max_user_instances</td>
<td>1048</td>
<td>128</td>
<td>Upper limit on the number of inotify instances that can be created per real user ID (see inotify)</td>
</tr>
<tr>
<td>fs.inotify.max_user_watches</td>
<td>1048</td>
<td>8192</td>
<td>Upper limit on the number of watches that can be created per real user ID (see inotify)</td>
</tr>
<tr>
<td>kernel.dmesg_restrict</td>
<td>1</td>
<td>0</td>
<td>Whether to deny container access to the messages in the kernel ring buffer (note that this will also deny access to non-root users on the host system)</td>
</tr>
<tr>
<td>kernel.keys.maxbytes</td>
<td>20000</td>
<td>2000</td>
<td>Maximum size of the key ring that non-root users can use</td>
</tr>
<tr>
<td>kernel.keys.maxkeys</td>
<td>20000</td>
<td>200</td>
<td>Maximum number of keys that a non-root user can use (the value should be higher than the number of instances)</td>
</tr>
<tr>
<td>net.core.bpf_jit_limit</td>
<td>100%</td>
<td>var</td>
<td>Limit on the size of eBPF JIT allocations (on kernels &lt; 5.15 that are compiled with CONFIG_BPF_JIT_ALWAYS_ON=y, this value might limit the amount of instances that can be created)</td>
</tr>
<tr>
<td>net.ipv4.neigh.default.gc_thresh3</td>
<td>8192</td>
<td>1024</td>
<td>Maximum number of entries in the IPv4 ARP table (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)</td>
</tr>
<tr>
<td>net.ipv6.neigh.default.gc_thresh3</td>
<td>8192</td>
<td>1024</td>
<td>Maximum number of entries in IPv6 ARP table (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)</td>
</tr>
<tr>
<td>vm.max_map_count</td>
<td>2621</td>
<td>6553</td>
<td>Maximum number of memory map areas a process may have (memory map areas are used as a side-effect of calling malloc, directly by mmap and mprotect, and also when loading shared libraries)</td>
</tr>
</tbody>
</table>

### Related topics

How-to guides:
- *How to benchmark performance*
- *How to increase the network bandwidth*
- *How to monitor metrics*

Explanation:
- *About performance tuning*
2.4.11 Project configuration

Projects can be configured through a set of key/value configuration options. See Configure a project for instructions on how to set these options.

The key/value configuration is namespaced. The following options are available:

- Project features
- Project limits
- Project restrictions
- Project-specific configuration

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

<table>
<thead>
<tr>
<th>Key:</th>
<th>features.images</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>Initial value:</td>
<td>true</td>
</tr>
</tbody>
</table>

This setting applies to both images and image aliases.

features.networks Whether to use a separate set of networks for the project

<table>
<thead>
<tr>
<th>Key:</th>
<th>features.networks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>Initial value:</td>
<td>false</td>
</tr>
</tbody>
</table>

features.networks.zones Whether to use a separate set of network zones for the project

<table>
<thead>
<tr>
<th>Key:</th>
<th>features.networks.zones</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>Initial value:</td>
<td>false</td>
</tr>
</tbody>
</table>

features.profiles Whether to use a separate set of profiles for the project
features.profiles

<table>
<thead>
<tr>
<th>Key:</th>
<th>features.profiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>Initial value:</td>
<td>true</td>
</tr>
</tbody>
</table>

features.storage.buckets Whether to use a separate set of storage buckets for the project

<table>
<thead>
<tr>
<th>Key:</th>
<th>features.storage.buckets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>Initial value:</td>
<td>true</td>
</tr>
</tbody>
</table>

features.storage.volumes Whether to use a separate set of storage volumes for the project

<table>
<thead>
<tr>
<th>Key:</th>
<th>features.storage.volumes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>Initial value:</td>
<td>true</td>
</tr>
</tbody>
</table>

**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 or limits.networks) or to the aggregate value of resource usage for all instances in the project (for example, limits.cpu or limits.processes). In the latter case, the limit usually applies to the Resource limits 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 configuration to 50GiB, the sum of the individual values of all limits.memory configuration keys defined on the project's instances will be kept under 50 GiB.

Similarly, setting the project's limits.cpu configuration key to 100 means that the sum of individual limits.cpu 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 for the instance configuration options.
- The limits.cpu configuration cannot be used if CPU pinning is enabled. This means that to use limits.cpu on a project, the limits.cpu 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 configuration must be set to an absolute value, not a percentage.

limits.containers Maximum number of containers that can be created in the project

<table>
<thead>
<tr>
<th>Key:</th>
<th>limits.containers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>integer</td>
</tr>
</tbody>
</table>
limits.cpu Maximum number of CPUs to use in the project

<table>
<thead>
<tr>
<th>Key</th>
<th>limits.cpu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>integer</td>
</tr>
</tbody>
</table>

This value is the maximum value for the sum of the individual *limits.cpu* configurations set on the instances of the project.

limits.disk Maximum disk space used by the project

<table>
<thead>
<tr>
<th>Key</th>
<th>limits.disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
</tbody>
</table>

This value is the maximum value of the aggregate disk space used by all instance volumes, custom volumes, and images of the project.

limits.instances Maximum number of instances that can be created in the project

<table>
<thead>
<tr>
<th>Key</th>
<th>limits.instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>integer</td>
</tr>
</tbody>
</table>

limits.memory Usage limit for the host's memory for the project

<table>
<thead>
<tr>
<th>Key</th>
<th>limits.memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
</tbody>
</table>

The value is the maximum value for the sum of the individual *limits.memory* configurations set on the instances of the project.

limits.networks Maximum number of networks that the project can have

<table>
<thead>
<tr>
<th>Key</th>
<th>limits.networks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>integer</td>
</tr>
</tbody>
</table>

limits.processes Maximum number of processes within the project

<table>
<thead>
<tr>
<th>Key</th>
<th>limits.processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>integer</td>
</tr>
</tbody>
</table>

This value is the maximum value for the sum of the individual *limits.processes* configurations set on the instances of the project.

limits.virtual-machines Maximum number of VMs that can be created in the project

<table>
<thead>
<tr>
<th>Key</th>
<th>limits.virtual-machines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>integer</td>
</tr>
</tbody>
</table>
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` configuration option to `true`. You can then use the various `restricted.*` options to pick individual features that would normally be blocked by `restricted` 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

<table>
<thead>
<tr>
<th>Key</th>
<th>restricted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>false</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Key</th>
<th>restricted.backups</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>block</td>
</tr>
</tbody>
</table>

Possible values are `allow` or `block`.

**restricted.cluster.groups** Cluster groups that can be targeted

<table>
<thead>
<tr>
<th>Key</th>
<th>restricted.cluster.groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
</tbody>
</table>

If specified, this option prevents targeting cluster groups other than the provided ones.

**restricted.cluster.target** Whether to prevent targeting of cluster members

<table>
<thead>
<tr>
<th>Key</th>
<th>restricted.cluster.target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>block</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Key:</th>
<th>restricted.containers.interception</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>block</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Key:</th>
<th>restricted.containers.lowlevel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>block</td>
</tr>
</tbody>
</table>

Possible values are allow or block. When set to allow, low-level container options like `raw.lxc`, `raw.idmap`, `volatile.*`, etc. can be used.

**restricted.containers.nesting** Whether to prevent running nested LXD

<table>
<thead>
<tr>
<th>Key:</th>
<th>restricted.containers.nesting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>block</td>
</tr>
</tbody>
</table>

Possible values are allow or block. When set to allow, `security.nesting` can be set to `true` for an instance.

**restricted.containers.privilege** Which settings for privileged containers to prevent

<table>
<thead>
<tr>
<th>Key:</th>
<th>restricted.containers.privilege</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>unprivileged</td>
</tr>
</tbody>
</table>

Possible values are unprivileged, isolated, and allow.

- When set to unprivileged, this option prevents setting `security.privileged` to `true`.
- When set to isolated, this option prevents setting `security.privileged` and `security.idmap.isolated` to `true`.
- When set to allow, there is no restriction.

**restricted.devices.disk** Which disk devices can be used

<table>
<thead>
<tr>
<th>Key:</th>
<th>restricted.devices.disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>managed</td>
</tr>
</tbody>
</table>

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.

restricted.devices.disk.paths Which source can be used for disk devices

<table>
<thead>
<tr>
<th>Key</th>
<th>restricted.devices.disk.paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
</tbody>
</table>

If restricted.devices.disk 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

<table>
<thead>
<tr>
<th>Key</th>
<th>restricted.devices.gpu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>block</td>
</tr>
</tbody>
</table>

Possible values are allow or block.

restricted.devices.infiniband Whether to prevent using devices of type infiniband

<table>
<thead>
<tr>
<th>Key</th>
<th>restricted.devices.infiniband</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>block</td>
</tr>
</tbody>
</table>

Possible values are allow or block.

restricted.devices.nic Which network devices can be used

<table>
<thead>
<tr>
<th>Key</th>
<th>restricted.devices.nic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>managed</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Key</th>
<th>restricted.devices.pci</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>block</td>
</tr>
</tbody>
</table>

Possible values are allow or block.

restricted.devices.proxy Whether to prevent using devices of type proxy

2.4. Reference
Canalเทศ LXD

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>restricted.devices.proxy</td>
<td>string</td>
<td>block</td>
</tr>
</tbody>
</table>

Possible values are allow or block.

restricted.devices.unix-block Whether to prevent using devices of type unix-block

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>restricted.devices.unix-block</td>
<td>string</td>
<td>block</td>
</tr>
</tbody>
</table>

Possible values are allow or block.

restricted.devices.unix-char Whether to prevent using devices of type unix-char

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>restricted.devices.unix-char</td>
<td>string</td>
<td>block</td>
</tr>
</tbody>
</table>

Possible values are allow or block.

restricted.devices.unix-hotplug Whether to prevent using devices of type unix-hotplug

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>restricted.devices.unix-hotplug</td>
<td>string</td>
<td>block</td>
</tr>
</tbody>
</table>

Possible values are allow or block.

restricted.devices.usb Whether to prevent using devices of type usb

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>restricted.devices.usb</td>
<td>string</td>
<td>block</td>
</tr>
</tbody>
</table>

Possible values are allow or block.

restricted.idmap.gid Which host GID ranges are allowed in raw.idmap

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>restricted.idmap.gid</td>
<td>string</td>
</tr>
</tbody>
</table>

This option specifies the host GID ranges that are allowed in the instance’s raw.idmap setting.

restricted.idmap.uid Which host UID ranges are allowed in raw.idmap

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>restricted.idmap.uid</td>
<td>string</td>
</tr>
</tbody>
</table>
This option specifies the host UID ranges that are allowed in the instance’s `raw.idmap` setting.

**restricted.networks.access** Which network names are allowed for use in this project

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>restricted.networks.access</td>
<td>string</td>
</tr>
</tbody>
</table>

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` setting.

**restricted.networks.subnets** Which network subnets are allocated for use in this project

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>restricted.networks.subnets</td>
<td>string</td>
<td>block</td>
</tr>
</tbody>
</table>

Specify a comma-delimited list of network subnets from the uplink networks that are allocated for use in this project. Use the form `<uplink>:<subnet>`.

**restricted.networks.uplinks** Which network names can be used as uplink in this project

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>restricted.networks.uplinks</td>
<td>string</td>
<td>block</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>restricted.networks.zones</td>
<td>string</td>
<td>block</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>restricted.snapshots</td>
<td>string</td>
<td>block</td>
</tr>
</tbody>
</table>

**restricted.virtual-machines.lowlevel** Whether to prevent using low-level VM options

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>restricted.virtual-machines.lowlevel</td>
<td>string</td>
<td>block</td>
</tr>
</tbody>
</table>

Possible values are `allow` or `block`. When set to `allow`, low-level VM options like `raw.qemu`, `volatile.*`, etc. can be used.
Project-specific configuration

There are some Server configuration 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

<table>
<thead>
<tr>
<th>Key</th>
<th>backups.compression_algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Key</th>
<th>images.auto_update_cached</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
</tbody>
</table>

images.auto_update_interval Interval at which to look for updates to cached images

<table>
<thead>
<tr>
<th>Key</th>
<th>images.auto_update_interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>integer</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Key</th>
<th>images.compression_algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
</tbody>
</table>

Possible values are bzip2, gzip, lzma, xz, or none.

images.default_architecture Default architecture to use in a mixed-architecture cluster

<table>
<thead>
<tr>
<th>Key</th>
<th>images.default_architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
</tbody>
</table>

images.remote_cache_expiry When an unused cached remote image is flushed in the project

<table>
<thead>
<tr>
<th>Key</th>
<th>images.remote_cache_expiry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>integer</td>
</tr>
</tbody>
</table>

Specify the number of days after which the unused cached image expires.

user.* User-provided free-form key/value pairs

<table>
<thead>
<tr>
<th>Key</th>
<th>user.*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
</tbody>
</table>
2.4.12 Provided metrics

LXD provides a number of instance metrics and internal metrics. See *How to monitor metrics* for instructions on how to work with these metrics.

**Instance metrics**

The following instance metrics are provided:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lxd_cpu_effective_total</code></td>
<td>Total number of effective CPUs</td>
</tr>
<tr>
<td><code>lxd_cpu_seconds_total{cpu=&quot;&lt;cpu&gt;&quot;, mode=&quot;&lt;mode&gt;&quot;}</code></td>
<td>Total number of CPU time used (in seconds)</td>
</tr>
<tr>
<td><code>lxd_disk_read_bytes_total{device=&quot;&lt;dev&gt;&quot;}</code></td>
<td>Total number of bytes read</td>
</tr>
<tr>
<td><code>lxd_disk_reads_completed_total{device=&quot;&lt;dev&gt;&quot;}</code></td>
<td>Total number of completed reads</td>
</tr>
<tr>
<td><code>lxd_disk_written_bytes_total{device=&quot;&lt;dev&gt;&quot;}</code></td>
<td>Total number of bytes written</td>
</tr>
<tr>
<td><code>lxd_disk_writes_completed_total{device=&quot;&lt;dev&gt;&quot;}</code></td>
<td>Total number of completed writes</td>
</tr>
<tr>
<td><code>lxd_disk_written_bytes{device=&quot;&lt;dev&gt;&quot;,fstype=&quot;&lt;type&gt;&quot;}</code></td>
<td>Available space (in bytes)</td>
</tr>
<tr>
<td><code>lxd_disk_read_bytes{device=&quot;&lt;dev&gt;&quot;,fstype=&quot;&lt;type&gt;&quot;}</code></td>
<td>Free space (in bytes)</td>
</tr>
<tr>
<td><code>lxd_disk_size_bytes{device=&quot;&lt;dev&gt;&quot;,fstype=&quot;&lt;type&gt;&quot;}</code></td>
<td>Size of the file system (in bytes)</td>
</tr>
<tr>
<td><code>lxd_memory_Active_anon_bytes</code></td>
<td>Amount of anonymous memory on active LRU list</td>
</tr>
<tr>
<td><code>lxd_memory_Active_bytes</code></td>
<td>Amount of memory on active LRU list</td>
</tr>
<tr>
<td><code>lxd_memory_File_bytes</code></td>
<td>Amount of file-backed memory on active LRU list</td>
</tr>
<tr>
<td><code>lxd_memory_Cached_bytes</code></td>
<td>Amount of cached memory</td>
</tr>
<tr>
<td><code>lxd_memory_Dirty_bytes</code></td>
<td>Amount of memory waiting to be written back to the disk</td>
</tr>
<tr>
<td><code>lxd_memory_HugepagesFree_bytes</code></td>
<td>Amount of free memory for hugetlb</td>
</tr>
<tr>
<td><code>lxd_memory_HugepagesTotal_bytes</code></td>
<td>Amount of used memory for hugetlb</td>
</tr>
<tr>
<td><code>lxd_memory_Inactive_anon_bytes</code></td>
<td>Amount of anonymous memory on inactive LRU list</td>
</tr>
<tr>
<td><code>lxd_memory_Inactive_bytes</code></td>
<td>Amount of memory on inactive LRU list</td>
</tr>
<tr>
<td><code>lxd_memory_Inactive_file_bytes</code></td>
<td>Amount of file-backed memory on inactive LRU list</td>
</tr>
<tr>
<td><code>lxd_memory_Mapped_bytes</code></td>
<td>Amount of mapped memory</td>
</tr>
<tr>
<td><code>lxd_memory_MemAvailable_bytes</code></td>
<td>Amount of available memory</td>
</tr>
<tr>
<td><code>lxd_memory_MemFree_bytes</code></td>
<td>Amount of free memory</td>
</tr>
<tr>
<td><code>lxd_memory_MemTotal_bytes</code></td>
<td>Amount of used memory</td>
</tr>
<tr>
<td><code>lxd_memory_oom_kills_total</code></td>
<td>The number of out-of-memory kills</td>
</tr>
<tr>
<td><code>lxd_memory_RSS_bytes</code></td>
<td>Amount of anonymous and swap cache memory</td>
</tr>
<tr>
<td><code>lxd_memory_Shmem_bytes</code></td>
<td>Amount of cached file system data that is swap-backed</td>
</tr>
<tr>
<td><code>lxd_memory_Swap_bytes</code></td>
<td>Amount of swap memory</td>
</tr>
<tr>
<td><code>lxd_memory_Unevictable_bytes</code></td>
<td>Amount of unevictable memory</td>
</tr>
<tr>
<td><code>lxd_memory_Writeback_bytes</code></td>
<td>Amount of memory queued for syncing to disk</td>
</tr>
<tr>
<td><code>lxd_network_receive_bytes_total{device=&quot;&lt;dev&gt;&quot;}</code></td>
<td>Amount of received bytes on a given interface</td>
</tr>
<tr>
<td><code>lxd_network_receive_drop_total{device=&quot;&lt;dev&gt;&quot;}</code></td>
<td>Amount of received dropped bytes on a given interface</td>
</tr>
<tr>
<td><code>lxd_network_receive_errs_total{device=&quot;&lt;dev&gt;&quot;}</code></td>
<td>Amount of received errors on a given interface</td>
</tr>
</tbody>
</table>

*continues on next page*
Table 2 – continued from previous page

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lxd_network_receive_packets_total{device=&quot;&lt;dev&gt;&quot;}</td>
<td>Amount of received packets on a given interface</td>
</tr>
<tr>
<td>lxd_network_transmit_bytes_total{device=&quot;&lt;dev&gt;&quot;}</td>
<td>Amount of transmitted bytes on a given interface</td>
</tr>
<tr>
<td>lxd_network_transmit_drop_total{device=&quot;&lt;dev&gt;&quot;}</td>
<td>Amount of transmitted dropped bytes on a given interface</td>
</tr>
<tr>
<td>lxd_network_transmit_errs_total{device=&quot;&lt;dev&gt;&quot;}</td>
<td>Amount of transmitted errors on a given interface</td>
</tr>
<tr>
<td>lxd_network_transmit_packets_total{device=&quot;&lt;dev&gt;&quot;}</td>
<td>Amount of transmitted packets on a given interface</td>
</tr>
<tr>
<td>lxd_procs_total</td>
<td>Number of running processes</td>
</tr>
</tbody>
</table>

**Internal metrics**

The following internal metrics are provided:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lxd_go_alloc_bytes_total</td>
<td>Total number of bytes allocated (even if freed)</td>
</tr>
<tr>
<td>lxd_go_alloc_bytes</td>
<td>Number of bytes allocated and still in use</td>
</tr>
<tr>
<td>lxd_go_buck_hash_sys_bytes</td>
<td>Number of bytes used by the profiling bucket hash table</td>
</tr>
<tr>
<td>lxd_go_frees_total</td>
<td>Total number of frees</td>
</tr>
<tr>
<td>lxd_go_gc_sys_bytes</td>
<td>Number of bytes used for garbage collection system metadata</td>
</tr>
<tr>
<td>lxd_go_goroutines</td>
<td>Number of goroutines that currently exist</td>
</tr>
<tr>
<td>lxd_go_heap_alloc_bytes</td>
<td>Number of heap bytes allocated and still in use</td>
</tr>
<tr>
<td>lxd_go_heap_idle_bytes</td>
<td>Number of heap bytes waiting to be used</td>
</tr>
<tr>
<td>lxd_go_heap_inuse_bytes</td>
<td>Number of heap bytes that are in use</td>
</tr>
<tr>
<td>lxd_go_heap_objects</td>
<td>Number of allocated objects</td>
</tr>
<tr>
<td>lxd_go_heap_released_bytes</td>
<td>Number of heap bytes released to OS</td>
</tr>
<tr>
<td>lxd_go_heap_sys_bytes</td>
<td>Number of heap bytes obtained from system</td>
</tr>
<tr>
<td>lxd_go_lookups_total</td>
<td>Total number of pointer lookups</td>
</tr>
<tr>
<td>lxd_go_malloccs_total</td>
<td>Total number of mallocs</td>
</tr>
<tr>
<td>lxd_go_mcache_inuse_bytes</td>
<td>Number of bytes in use by mcache structures</td>
</tr>
<tr>
<td>lxd_go_mcache_sys_bytes</td>
<td>Number of bytes used for mcache structures obtained from system</td>
</tr>
<tr>
<td>lxd_go_mspan_inuse_bytes</td>
<td>Number of bytes in use by mspan structures</td>
</tr>
<tr>
<td>lxd_go_mspan_sys_bytes</td>
<td>Number of bytes used for mspan structures obtained from system</td>
</tr>
<tr>
<td>lxd_go_next_gc_bytes</td>
<td>Number of bytes used for mspan structures obtained from system</td>
</tr>
<tr>
<td>lxd_go_other_sys_bytes</td>
<td>Number of bytes used for other system allocations</td>
</tr>
<tr>
<td>lxd_go_stack_inuse_bytes</td>
<td>Number of bytes in use by the stack allocator</td>
</tr>
<tr>
<td>lxd_go_stack_sys_bytes</td>
<td>Number of bytes obtained from system for stack allocator</td>
</tr>
<tr>
<td>lxd_go_sys_bytes</td>
<td>Number of bytes obtained from system</td>
</tr>
<tr>
<td>lxd_operations_total</td>
<td>Number of running operations</td>
</tr>
<tr>
<td>lxd_uptime_seconds</td>
<td>Daemon uptime (in seconds)</td>
</tr>
<tr>
<td>lxd_warnings_total</td>
<td>Number of active warnings</td>
</tr>
</tbody>
</table>
2.4.13 Remote image servers

The `lxc` CLI command comes pre-configured with the following default remote image servers:

**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` 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` 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` 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` for an overview of available images.

**images:**
This server provides unofficial images for a variety of Linux distributions. The images are maintained by the Linux Containers team and are built to be compact and minimal.
See `images.linuxcontainers.org` 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. 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.
To make a LXD server publicly available over the network on port 8443, set the `core.https_address` configuration option to `:8443` and do not configure any authentication methods (see *How to expose LXD to the network* 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* and *Remote API authentication* for more information.

**Related topics**

How-to guides:

- *Images*

Explanation:

- *About images*

## 2.4.14 Requirements

**Go**

LXD requires Go 1.18 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 5.4.

LXD requires a kernel with support for:

- Namespaces (*pid, net, uts, ipc* and *mount*)
- Seccomp
- Native Linux AIO (*io_setup(2)*, etc.)

The following optional features also require extra kernel options:

- 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)

As well as any other kernel feature required by the LXC version in use.

**LXC**

LXD requires LXC 4.0.0 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.
QEMU

For virtual machines, QEMU 6.0 or higher is required.

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` >= 3.25.0 (for dqlite)

Make sure you have all these libraries themselves and their development headers (-dev packages) installed.

Related topics

Tutorials:
- First steps with LXD

How-to guides:
- Getting started

2.4.15 REST API

Events

Introduction

Events are messages about actions that have occurred over LXD. Using the API endpoint `/1.0/events` directly or via `lxc monitor` 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

```json
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>certificate-created</td>
<td>A new certificate has been added to the server trust store.</td>
<td></td>
</tr>
<tr>
<td>certificate-deleted</td>
<td>The certificate has been deleted from the trust store.</td>
<td></td>
</tr>
<tr>
<td>certificate-updated</td>
<td>The certificate’s configuration has been updated.</td>
<td></td>
</tr>
<tr>
<td>cluster-certificate-updated</td>
<td>The certificate for the whole cluster has changed.</td>
<td></td>
</tr>
<tr>
<td>cluster-disabled</td>
<td>Clustering has been disabled for this machine.</td>
<td></td>
</tr>
<tr>
<td>cluster-enabled</td>
<td>Clustering has been enabled for this machine.</td>
<td></td>
</tr>
<tr>
<td>cluster-group-created</td>
<td>A new cluster group has been created.</td>
<td></td>
</tr>
<tr>
<td>cluster-group-deleted</td>
<td>A cluster group has been deleted.</td>
<td></td>
</tr>
<tr>
<td>cluster-group-renamed</td>
<td>A cluster group has been renamed.</td>
<td></td>
</tr>
<tr>
<td>cluster-group-updated</td>
<td>A cluster group has been updated.</td>
<td></td>
</tr>
<tr>
<td>cluster-member-added</td>
<td>A new machine has joined the cluster.</td>
<td></td>
</tr>
<tr>
<td>cluster-member-removed</td>
<td>The cluster member has been removed from the cluster.</td>
<td>old_name</td>
</tr>
<tr>
<td>cluster-member-renamed</td>
<td>The cluster member has been renamed.</td>
<td>old_name</td>
</tr>
<tr>
<td>config-updated</td>
<td>The server configuration has changed.</td>
<td></td>
</tr>
<tr>
<td>image-alias-created</td>
<td>An alias has been created for an existing image.</td>
<td>target: the original instance.</td>
</tr>
<tr>
<td>image-alias-deleted</td>
<td>An alias has been deleted for an existing image.</td>
<td>target: the original instance.</td>
</tr>
<tr>
<td>image-alias-renamed</td>
<td>The alias for an existing image has been renamed.</td>
<td>old_name</td>
</tr>
<tr>
<td>image-alias-updated</td>
<td>The configuration for an image alias has changed.</td>
<td>target: the original instance.</td>
</tr>
<tr>
<td>image-created</td>
<td>A new image has been added to the image store.</td>
<td>type: container or disk image.</td>
</tr>
<tr>
<td>image-deleted</td>
<td>The image has been deleted from the image store.</td>
<td></td>
</tr>
<tr>
<td>image-refreshed</td>
<td>The local image copy has updated to the current source image version.</td>
<td></td>
</tr>
<tr>
<td>image-retrieved</td>
<td>The raw image file has been downloaded from the server.</td>
<td>target: destination server.</td>
</tr>
<tr>
<td>image-secret-created</td>
<td>A one-time key to fetch this image has been created.</td>
<td></td>
</tr>
<tr>
<td>image-updated</td>
<td>The image’s configuration has changed.</td>
<td></td>
</tr>
<tr>
<td>instance-backup-created</td>
<td>A backup of the instance has been created.</td>
<td></td>
</tr>
<tr>
<td>instance-backup-deleted</td>
<td>The instance backup has been deleted.</td>
<td></td>
</tr>
<tr>
<td>instance-backup-renamed</td>
<td>The instance backup has been renamed.</td>
<td>old_name</td>
</tr>
<tr>
<td>instance-backup-retrieved</td>
<td>The raw instance backup file has been downloaded.</td>
<td></td>
</tr>
<tr>
<td>instance-console</td>
<td>Connected to the console of the instance.</td>
<td>type: console.</td>
</tr>
<tr>
<td>instance-console-reset</td>
<td>The console buffer has been reset.</td>
<td></td>
</tr>
<tr>
<td>instance-console-retrieved</td>
<td>The console log has been downloaded.</td>
<td></td>
</tr>
<tr>
<td>instance-created</td>
<td>A new instance has been created.</td>
<td></td>
</tr>
<tr>
<td>instance-deleted</td>
<td>The instance has been deleted.</td>
<td></td>
</tr>
<tr>
<td>instance-exec</td>
<td>A command has been executed on the instance.</td>
<td>command: the command to be executed.</td>
</tr>
<tr>
<td>instance-file-deleted</td>
<td>A file on the instance has been deleted.</td>
<td>file: path.</td>
</tr>
<tr>
<td>instance-file-pushed</td>
<td>The file has been pushed to the instance.</td>
<td>file-source.</td>
</tr>
<tr>
<td>instance-file-retrieved</td>
<td>The file has been downloaded from the instance.</td>
<td>file-source.</td>
</tr>
</tbody>
</table>

2.4. Reference
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance-log-deleted</td>
<td>The instance’s specified log file has been deleted.</td>
<td></td>
</tr>
<tr>
<td>instance-log-retrieved</td>
<td>The instance’s specified log file has been downloaded.</td>
<td></td>
</tr>
<tr>
<td>instance-metadata-retrieved</td>
<td>The instance’s image metadata has been downloaded.</td>
<td></td>
</tr>
<tr>
<td>instance-metadata-template-created</td>
<td>A new image template file for the instance has been created.</td>
<td>path: relative file path.</td>
</tr>
<tr>
<td>instance-metadata-template-deleted</td>
<td>The image template file for the instance has been deleted.</td>
<td>path: relative file path.</td>
</tr>
<tr>
<td>instance-metadata-template-retrieved</td>
<td>The instance’s image metadata has been downloaded.</td>
<td></td>
</tr>
<tr>
<td>instance-paused</td>
<td>The instance has been put in a paused state.</td>
<td></td>
</tr>
<tr>
<td>instance-ready</td>
<td>The instance is ready.</td>
<td></td>
</tr>
<tr>
<td>instance-renamed</td>
<td>The instance has been renamed.</td>
<td>old_name</td>
</tr>
<tr>
<td>instance-restarted</td>
<td>The instance has restarted.</td>
<td></td>
</tr>
<tr>
<td>instance-restored</td>
<td>The instance has been restored from a snapshot.</td>
<td>snapshot</td>
</tr>
<tr>
<td>instance-resumed</td>
<td>The instance has resumed after being paused.</td>
<td></td>
</tr>
<tr>
<td>instance-shutdown</td>
<td>The instance has shut down.</td>
<td></td>
</tr>
<tr>
<td>instance-snapshot-created</td>
<td>A snapshot of the instance has been created.</td>
<td></td>
</tr>
<tr>
<td>instance-snapshot-deleted</td>
<td>The instance snapshot has been deleted.</td>
<td></td>
</tr>
<tr>
<td>instance-snapshot-renamed</td>
<td>The instance snapshot has been renamed.</td>
<td>old_name</td>
</tr>
<tr>
<td>instance-snapshot-updated</td>
<td>The instance snapshot’s configuration has changed.</td>
<td></td>
</tr>
<tr>
<td>instance-started</td>
<td>The instance has started.</td>
<td></td>
</tr>
<tr>
<td>instance-stopped</td>
<td>The instance has stopped.</td>
<td></td>
</tr>
<tr>
<td>instance-updated</td>
<td>The instance’s configuration has changed.</td>
<td></td>
</tr>
<tr>
<td>network-acl-created</td>
<td>A new network ACL has been created.</td>
<td></td>
</tr>
<tr>
<td>network-acl-deleted</td>
<td>The network ACL has been deleted.</td>
<td></td>
</tr>
<tr>
<td>network-acl-renamed</td>
<td>The network ACL has been renamed.</td>
<td>old_name</td>
</tr>
<tr>
<td>network-acl-updated</td>
<td>The network ACL configuration has changed.</td>
<td></td>
</tr>
<tr>
<td>network-created</td>
<td>A network device has been created.</td>
<td></td>
</tr>
<tr>
<td>network-deleted</td>
<td>The network device has been deleted.</td>
<td></td>
</tr>
<tr>
<td>network-forward-created</td>
<td>A new network forward has been created.</td>
<td></td>
</tr>
<tr>
<td>network-forward-deleted</td>
<td>The network forward has been deleted.</td>
<td></td>
</tr>
<tr>
<td>network-forward-updated</td>
<td>The network forward has been updated.</td>
<td></td>
</tr>
<tr>
<td>network-peer-created</td>
<td>A new network peer has been created.</td>
<td></td>
</tr>
<tr>
<td>network-peer-deleted</td>
<td>The network peer has been deleted.</td>
<td></td>
</tr>
<tr>
<td>network-peer-updated</td>
<td>The network peer has been updated.</td>
<td></td>
</tr>
<tr>
<td>network-renamed</td>
<td>The network device has been renamed.</td>
<td>old_name</td>
</tr>
<tr>
<td>network-updated</td>
<td>The network device’s configuration has changed.</td>
<td></td>
</tr>
<tr>
<td>network-zone-created</td>
<td>A new network zone has been created.</td>
<td></td>
</tr>
<tr>
<td>network-zone-deleted</td>
<td>The network zone has been deleted.</td>
<td></td>
</tr>
<tr>
<td>network-zone-record-created</td>
<td>A new network zone record has been created.</td>
<td></td>
</tr>
<tr>
<td>network-zone-record-deleted</td>
<td>The network zone record has been created.</td>
<td></td>
</tr>
<tr>
<td>network-zone-record-updated</td>
<td>The network zone record has been updated.</td>
<td></td>
</tr>
<tr>
<td>network-zone-updated</td>
<td>The network zone has been updated.</td>
<td></td>
</tr>
<tr>
<td>operation-cancelled</td>
<td>The operation has been canceled.</td>
<td></td>
</tr>
<tr>
<td>profile-created</td>
<td>A new profile has been created.</td>
<td></td>
</tr>
<tr>
<td>profile-deleted</td>
<td>The profile has been deleted.</td>
<td></td>
</tr>
<tr>
<td>profile-renamed</td>
<td>The profile has been renamed.</td>
<td>old_name</td>
</tr>
<tr>
<td>profile-updated</td>
<td>The profile’s configuration has changed.</td>
<td></td>
</tr>
<tr>
<td>project-created</td>
<td>A new project has been created.</td>
<td></td>
</tr>
<tr>
<td>project-deleted</td>
<td>The project has been deleted.</td>
<td></td>
</tr>
<tr>
<td>project-renamed</td>
<td>The project has been renamed.</td>
<td>old_name</td>
</tr>
<tr>
<td>project-updated</td>
<td>The project’s configuration has changed.</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Additional</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>storage-pool-created</td>
<td>A new storage pool has been created.</td>
<td></td>
</tr>
<tr>
<td>storage-pool-deleted</td>
<td>The storage pool has been deleted.</td>
<td></td>
</tr>
<tr>
<td>storage-pool-updated</td>
<td>The storage pool’s configuration has changed.</td>
<td></td>
</tr>
<tr>
<td>storage-volume-backup-created</td>
<td>A new backup for the storage volume has been created.</td>
<td></td>
</tr>
<tr>
<td>storage-volume-backup-deleted</td>
<td>The storage volume’s backup has been deleted.</td>
<td></td>
</tr>
<tr>
<td>storage-volume-backup-renamed</td>
<td>The storage volume’s backup has been renamed.</td>
<td>old_name</td>
</tr>
<tr>
<td>storage-volume-backup-retrieved</td>
<td>The storage volume’s backup has been downloaded.</td>
<td></td>
</tr>
<tr>
<td>storage-volume-created</td>
<td>A new storage volume has been created.</td>
<td>type: container, virtual-machine, image, custom</td>
</tr>
<tr>
<td>storage-volume-deleted</td>
<td>The storage volume has been deleted.</td>
<td></td>
</tr>
<tr>
<td>storage-volume-renamed</td>
<td>The storage volume has been renamed.</td>
<td>old_name</td>
</tr>
<tr>
<td>storage-volume-restored</td>
<td>The storage volume has been restored from a snapshot.</td>
<td>snapshot</td>
</tr>
<tr>
<td>storage-volume-snapshot-created</td>
<td>A new storage volume snapshot has been created.</td>
<td>type: container, virtual-machine, image, custom</td>
</tr>
<tr>
<td>storage-volume-snapshot-deleted</td>
<td>The storage volume’s snapshot has been deleted.</td>
<td></td>
</tr>
<tr>
<td>storage-volume-snapshot-renamed</td>
<td>The storage volume’s snapshot has been renamed.</td>
<td>old_name</td>
</tr>
<tr>
<td>storage-volume-snapshot-updated</td>
<td>The configuration for the storage volume’s snapshot has changed.</td>
<td></td>
</tr>
<tr>
<td>storage-volume-updated</td>
<td>The storage volume’s configuration has changed.</td>
<td></td>
</tr>
<tr>
<td>warning-acknowledged</td>
<td>The warning’s status has been set to “acknowledged”.</td>
<td></td>
</tr>
<tr>
<td>warning-deleted</td>
<td>The warning has been deleted.</td>
<td></td>
</tr>
<tr>
<td>warning-reset</td>
<td>The warning’s status has been set to “new”.</td>
<td></td>
</tr>
</tbody>
</table>

**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` 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.
**Canonical LXD**

**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:

```json
[
  "/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:

```json
{
  "eth0": {
    "name": "eth0",
    "network": "lxdb0",
    "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", "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.dev.lxd.images 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:

```yaml
#cloud-config
instance-id: af6a01c7-f847-4688-a2a4-37fddd744625
local-hostname: abc
```
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 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) 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 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:

```json
{
   "type": "sync",
   "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": {
        "containers": [  // Dictionary of resource types (container, snapshots, images) and affected resources
            "/1.0/instances/test"
        ]
    },
    "metadata": {},  // Metadata specific to the operation in question (in this case, exec)
    "fds": {  // Fds
        "0": "2a4a97af81529f6608dca31f03a7b7e47acc0b8dc6514496eb25e325f9e4fa6a",
        "control": "5b64c661ef313b423b5317ba9cb6410e40b705806c28255f601c0ef603f079a7"
    },
    "may_cancel": false,  // Whether the operation can be canceled (DELETE over REST)
    "err": ""  // The error string should be empty if 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

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Operation created</td>
</tr>
<tr>
<td>101</td>
<td>Started</td>
</tr>
<tr>
<td>102</td>
<td>Stopped</td>
</tr>
<tr>
<td>103</td>
<td>Running</td>
</tr>
<tr>
<td>104</td>
<td>Canceling</td>
</tr>
<tr>
<td>105</td>
<td>Pending</td>
</tr>
<tr>
<td>106</td>
<td>Starting</td>
</tr>
<tr>
<td>107</td>
<td>Stopping</td>
</tr>
<tr>
<td>108</td>
<td>Aborting</td>
</tr>
<tr>
<td>109</td>
<td>Freezing</td>
</tr>
<tr>
<td>110</td>
<td>Frozen</td>
</tr>
<tr>
<td>111</td>
<td>Thawed</td>
</tr>
<tr>
<td>112</td>
<td>Error</td>
</tr>
<tr>
<td>113</td>
<td>Ready</td>
</tr>
<tr>
<td>200</td>
<td>Success</td>
</tr>
<tr>
<td>400</td>
<td>Failure</td>
</tr>
<tr>
<td>401</td>
<td>Cancelled</td>
</tr>
</tbody>
</table>

**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:
Here are a few GET query examples of the different filtering methods mentioned above:

<table>
<thead>
<tr>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>?filter=devices.device_name.field_name eq desired_field_assignment</td>
</tr>
<tr>
<td>containers?filter=name eq &quot;my container&quot; and status eq Running</td>
</tr>
<tr>
<td>containers?filter=config.image.os eq ubuntu or devices.eth0.nictype eq bridged</td>
</tr>
<tr>
<td>images?filter=Properties.os eq Centos and not UpdateSource.Protocol eq simplesstreams</td>
</tr>
</tbody>
</table>

**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.

PUT replaces the entire object with a new definition, it’s typically called after the current object state was retrieved 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.

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 specification describing its API endpoints. The YAML version of this API specification can be found in rest-api.yaml. See Main API specification for a convenient web rendering of it.

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 storage.zfs_remove_snapshots 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 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 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.blacklist_default
- security.syscalls.blacklist_compat
- security.syscalls.blacklist
- security.syscalls.whitelist

See Instance configuration for how to use them.
Canonical LXD

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 About security 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 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=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).

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 storage.zfs_use_refquota 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.

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 for details)
- POST /1.0/networks (see RESTful API for details)
- PUT /1.0/networks/<entry> (see RESTful API for details)
- PATCH /1.0/networks/<entry> (see RESTful API for details)
- DELETE /1.0/networks/<entry> (see RESTful API 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` and `security.idmap.isolated`, `security.idmap.size`, and `raw.id_map` fields.

network_firewall_filtering

Add two new keys, `ipv4.firewall` and `ipv6.firewall` 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` or `ipv6.nat` 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` and `ipv6.routes` 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 for details)
- GET /1.0/storage-pools/<name> (see RESTful API for details)
- POST /1.0/storage-pools/<name> (see RESTful API for details)
- PUT /1.0/storage-pools/<name> (see RESTful API for details)
- PATCH /1.0/storage-pools/<name> (see RESTful API for details)
- DELETE /1.0/storage-pools/<name> (see RESTful API for details)
- GET /1.0/storage-pools/<name>/volumes (see RESTful API for details)
- GET /1.0/storage-pools/<name>/volumes/<volume_type> (see RESTful API for details)
- POST /1.0/storage-pools/<name>/volumes/<volume_type> (see RESTful API for details)
- GET /1.0/storage-pools/<pool>/volumes/<volume_type>/<name> (see RESTful API for details)
- POST /1.0/storage-pools/<pool>/volumes/<volume_type>/<name> (see RESTful API for details)
- PUT /1.0/storage-pools/<pool>/volumes/<volume_type>/<name> (see RESTful API for details)
- PATCH /1.0/storage-pools/<pool>/volumes/<volume_type>/<name> (see RESTful API for details)
- DELETE /1.0/storage-pools/<pool>/volumes/<volume_type>/<name> (see RESTful API for details)
- All storage configuration options (see Storage configuration 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 and ipv6.dhcp.expiry allowing to set the DHCP lease expiry time.

storage_lvm_vg_rename

Introduces the ability to rename a volume group by setting storage.lvm.vg_name.

storage_lvm_thinpool_rename

Introduces the ability to rename a thin pool name by setting storage.thinpool_name.

network_vlan

This adds a new vlan 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 `storage_zfs_clone_copy` 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` 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` 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 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 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.
macaroon_authentication

This adds support for external authentication via Macaroons.

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 container configuration key was introduced. The key controls whether the /dev/lxd interface is made available to the container. 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` 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` 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 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
• GET /1.0/networks accepts a new target query parameter
• POST /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 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 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 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` 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 \(1.2.3.4\ 0-1000 \rightarrow 5.6.7.8\ 1000-2000\)) and between a range of source ports and a single target port (for example \(1.2.3.4\ 0-1000 \rightarrow 5.6.7.8\ 1000\)).

network_state

Adds support for retrieving a network’s state.

This adds the following new endpoint (see RESTful API 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` field which prevents containers from being deleted if set to `true`. Snapshots are not affected by this setting.

proxy_priv_drop

Adds `security.uid` and `security.gid` 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` 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` key to the proxy device which controls the use of the HAProxy PROXY protocol header.
network_hwaddr

Adds a bridge.hwaddr 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 and ipv6.nat.order configuration keys for LXD bridges. Those keys control whether to put the LXD rules before or after any pre-existing rules in the chain.

candidate_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 to get all it needs in one query.

candid_authentication

This introduces the new candid.api.url configuration option and removes core.macaroon.endpoint.

backup_compression

This introduces a new backups.compression_algorithm configuration key which allows configuration of backup compression.

candid_config

This introduces the configuration keys candid.domains and candid.expiry. The former allows specifying allowed/valid Candid domains, the latter makes the macaroon's expiry configurable. The lxc remote add command now has a --domain flag which allows specifying a Candid domain.

nvidia_runtime_config

This introduces a few extra configuration keys when using nvidia.runtime and the libnvidia-container library. Those keys translate pretty much directly to the matching NVIDIA container environment variables:

- nvidia.driver.capabilities => NVIDIA_DRIVER_CAPABILITIES
- nvidia.require.cuda => NVIDIA_REQUIRE_CUDA
- nvidia.require.driver => 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 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.

candid_config_key

This introduces a new candid.api.key option which allows for setting the expected public key for the endpoint, allowing for safe use of a HTTP-only Candid server.

network_vxlan_ttl

This adds a new tunnel.NAME.ttl 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 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@@ and @@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 configuration key to specify the address of the initial server. When joining a new server, clients can set the core.https_address 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 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 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` 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`. 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 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 key which is used to track the current mapping for the container.

This effectively gives us:

- volatile.last_state.idmap => On-disk idmap
- volatile.idmap.current => Current kernel map
- volatile.idmap.next => 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. shiftfs).

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 and ipv6.nat.address 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 and ipv6.routes properties on nic type devices. This allows adding static routes on host to container’s NIC.

rbac

Adds support for RBAC (role based access control). This introduces new configuration keys:

- rbac.api.url
- rbac.api.key
- rbac.api.expiry
- rbac.agent.url
- rbac.agent.username
- rbac.agent.private_key
- rbac.agent.public_key
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 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) and MAC filtering (security.mac_filtering) 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 and security.ipv6_filtering) 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 property on disk devices which controls the use of the shiftfs 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 shiftfs 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` and `storage.backups_volume` 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).

ceph_data_pool_name

This adds support for an optional argument (ceph.osd.data_pool_name) 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, security.syscalls.intercept.mount.allowed, and security.syscalls.intercept.mount.shift 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 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
Add support for filtering the result of a GET request for instances and images.

instance_nic_network
Add 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` and `cluster.max_standby` 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` and `ipv6.gateway` 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` 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` 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` and `ipv6.host_address` 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` and `ipv6.gateway` 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 and ipv6.gateway 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 and ipv6.host_table 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 and ipv6.host_table 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 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 and ipv6.address 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 and ipv6.gateway 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`, `limits.egress` and `limits.max` for routed NICs.

instance_nic_bridged_vlan

This introduces the `vlan` and `vlan.tagged` 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
  • security.syscalls.deny_compat
  • security.syscalls.deny
  • security.syscalls.allow
**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` 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` 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` 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` 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` 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` - the OVS integration bridge to use.
- `network.ovn.northbound_connection` - the OVN northbound database connection string.
projects_networks

Adds the features.networks configuration key to projects and the ability for a project to hold networks.

projects_networks_restricted_uplinks

Adds the restricted.networks.uplinks 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 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 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 and ipv6.routes setting on physical networks that defines the external routes allowed to be used in child OVN networks in their ipv4.routes.external and ipv6.routes.external settings.
Introduces the restricted.networks.subnets 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 and ipv6.nat 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 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. It introduces the mdev 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` 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` and `ipv6.dhcp` settings for OVN networks.
Allows DHCP (and RA for IPv6) to be disabled. Defaults to on.

network_physical_routes_anycast

Adds `ipv4.routes.anycast` and `ipv6.routes.anycast` 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=routed`.

projects_limits_instances

Adds `limits.instances` 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 field for bridged NIC instances.

instance_bulk_state_change

Adds the following endpoint for bulk state change (see RESTful API 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` configuration key.

disk_state_quota

This introduces the `size.state` device configuration key on disk devices.

storage_ceph_features

Adds a new `ceph.rbd.features` configuration key on storage pools to control the RBD features used for new volumes.

projects_compression

Adds new `backups.compression_algorithm` and `images.compression_algorithm` configuration keys which allows configuration of backup and image compression per-project.

projects_images_remote_cache_expiry

Add new `images.remote_cache_expiry` 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` and `images.auto_update_interval` configuration keys which allows configuration of images auto update in projects

projects_restricted_cluster_target

Adds new `restricted.cluster.target` 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.
Canonical LXD

**images_default_architecture**

Adds new `images.default_architecture` 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` 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* 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` and `restricted.snapshots` 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 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` and `volatile.evacuate.origin` 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` and `ipv6.nat.address` 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=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`
- `core.bgp_asn`
- `core.bgp_routerid`

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`
- `bgp.ipv6.nexthop`

And the following NIC-specific configuration keys (bridged NIC type):

- `ipv4.routes.external`
- `ipv6.routes.external`

**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

Addsthe 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
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
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 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`
- `cloud-init.user-data`
- `cloud-init.network-config`

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`

ceph_rbd_du

Adds a new `ceph.rbd.du` 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` 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, 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.

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 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 and ipv6.neighbor_probe 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.container.intercept configuration key to allow usually safe system call interception options.

metrics_authentication

Introduces a new core.metrics_authentication 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 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 storage.thinpool_metadata_size. 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/NLAME/file.

instances_nic_host_name

This introduces the instances.nic.host_name 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 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.

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 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 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 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 setting, described below).

This also introduces a change whereby network access is controlled by the project’s restricted.devices.nic 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 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: CA certificate which can be used when sending events to the Loki server
- loki.api.url: URL to the Loki server (protocol, name or IP and port)
- loki.auth.username and loki.auth.password: Used if Loki is behind a reverse proxy with basic authentication enabled
- loki.labels: Comma-separated list of values which are to be used as labels for Loki events.
- loki.loglevel: Minimum log level for events sent to the Loki server.
- loki.types: 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 or other ACME services to issue certificates.

It adds the following global configuration keys:

- acme.domain: The domain for which the certificate should be issued.
- acme.email: The email address used for the account of the ACME service.
- acme.ca_url: 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` configuration key.

**remote_token_expiry**

This adds an expiry to remote add join tokens. It can be set in the `core.remote_token_expiry` 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.

**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` 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`, meaning they were created in the same project as networks were.

Now this has been decoupled from `features.networks` 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_txqueue_length

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.

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
**disk_io_cache**

This introduces a new `io.cache` 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` : (bool) is SEV enabled for this VM
- `security.sev.policy.es` : (bool) is SEV-ES enabled for this VM
- `security.sev.session.dh` : (string) guest owner’s base64-encoded Diffie-Hellman key
- `security.sev.session.data` : (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.

**ovnNic_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` property and the VLAN ID to use for tunneling with the `vlan` property.

**oidc**

This adds support for OpenID Connect (OIDC) authentication.

This adds the following new configuration keys:

- `oidc.issuer`
- `oidc.client.id`
- `oidc.audience`
**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` and `ipv6.l3only` 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` 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`

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`, 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` 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`.

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`: (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`) 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`.

network_allocations

This adds the possibility to list a LXD deployment’s network allocations.

Through the `lxc network list-allocations` 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` 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 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 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.
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.

**Main API specification**

**Related topics**

How-to guides:

- LXD server and client

Explanation:

- About lxd and lxc
- About the LXD database

### 2.4.16 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:

- Core configuration
- ACME configuration
- Candid and RBAC configuration
- OpenID Connect configuration
- OpenFGA configuration
- Cluster configuration
- Images configuration
- Loki configuration
- Miscellaneous options

See *How to configure the LXD server* 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
  
<table>
<thead>
<tr>
<th>Key</th>
<th>core.bgp_address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Scope</td>
<td>local</td>
</tr>
</tbody>
</table>

  See *How to configure LXD as a BGP server*.

- **core.bgp_asn** BGP Autonomous System Number for the local server
  
<table>
<thead>
<tr>
<th>Key</th>
<th>core.bgp_asn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Scope</td>
<td>global</td>
</tr>
</tbody>
</table>

- **core.bgp_routerid** A unique identifier for the BGP server
  
<table>
<thead>
<tr>
<th>Key</th>
<th>core.bgp_routerid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Scope</td>
<td>local</td>
</tr>
</tbody>
</table>

  The identifier must be formatted as an IPv4 address.

- **core.debug_address** Address to bind the pprof debug server to (HTTP)
  
<table>
<thead>
<tr>
<th>Key</th>
<th>core.debug_address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Scope</td>
<td>local</td>
</tr>
</tbody>
</table>

- **core.dns_address** Address to bind the authoritative DNS server to
  
<table>
<thead>
<tr>
<th>Key</th>
<th>core.dns_address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Scope</td>
<td>local</td>
</tr>
</tbody>
</table>

  See *Enable the built-in DNS server*.

- **core.https_address** Address to bind for the remote API (HTTPS)
  
<table>
<thead>
<tr>
<th>Key</th>
<th>core.https_address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Scope</td>
<td>local</td>
</tr>
</tbody>
</table>

  See *How to expose LXD to the network*.

- **core.https_allowed_credentials** Whether to set Access-Control-Allow-Credentials
If enabled, the Access-Control-Allow-Credentials HTTP header value is set to true.

```
Key:    core.https_allowed_credentials
Type:   bool
Scope:  global
```

core.https_allowed_headers Access-Control-Allow-Headers HTTP header value

```
Key:    core.https_allowed_headers
Type:   string
Scope:  global
```

core.https_allowed_methods Access-Control-Allow-Methods HTTP header value

```
Key:    core.https_allowed_methods
Type:   string
Scope:  global
```

core.https_allowed_origin Access-Control-Allow-Origin HTTP header value

```
Key:    core.https_allowed_origin
Type:   string
Scope:  global
```

core.https_trusted_proxy Trusted servers to provide the client’s address

```
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)

```
Key:    core.metrics_address
Type:   string
Scope:  local
```

See *How to monitor metrics*.

core.metrics_authentication Whether to enforce authentication on the metrics endpoint

```
Key:    core.metrics_authentication
Type:   bool
Default: true
Scope:  global
```
core.proxy_http HTTP proxy to use

- **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

- **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

- **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

- **Key:** `core.remote_token_expiry`
- **Type:** `string`
- **Default:** `no expiry`
- **Scope:** `global`

core.shutdown_timeout How long to wait before shutdown

- **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)

- **Key:** `core.storage_buckets_address`
- **Type:** `string`
- **Scope:** `local`

See *How to manage storage buckets and keys*.

core.syslog_socket Whether to enable the syslog unixgram socket listener

2.4. Reference
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

<table>
<thead>
<tr>
<th>Key:</th>
<th>core.trust_ca_certificates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>bool</td>
</tr>
<tr>
<td>Scope:</td>
<td>global</td>
</tr>
</tbody>
</table>

### core.trust_password

Password to be provided by clients to set up a trust

<table>
<thead>
<tr>
<th>Key:</th>
<th>core.trust_password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Scope:</td>
<td>global</td>
</tr>
</tbody>
</table>

### ACME configuration

The following server options control the **ACME** configuration:

#### acme.agree_tos

Agree to ACME terms of service

<table>
<thead>
<tr>
<th>Key:</th>
<th>acme.agree_tos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>Scope:</td>
<td>global</td>
</tr>
</tbody>
</table>

#### acme.ca_url

Agree to ACME terms of service

<table>
<thead>
<tr>
<th>Key:</th>
<th>acme.ca_url</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td><a href="https://acme-v02.api.letsencrypt.org/directory">https://acme-v02.api.letsencrypt.org/directory</a></td>
</tr>
<tr>
<td>Scope:</td>
<td>global</td>
</tr>
</tbody>
</table>

#### acme.domain

Domain for which the certificate is issued

<table>
<thead>
<tr>
<th>Key:</th>
<th>acme.domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Scope:</td>
<td>global</td>
</tr>
</tbody>
</table>

#### acme.email

Email address used for the account registration

<table>
<thead>
<tr>
<th>Key:</th>
<th>acme.email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Scope:</td>
<td>global</td>
</tr>
</tbody>
</table>
Candid and RBAC configuration

The following server options configure external user authentication, through Candid-based authentication or through Role Based Access Control (RBAC):

- `candid.api_key` Public key of the Candid server

  | Key          | candid.api_key    | Type: string | Condition: required for HTTP-only servers | Scope: global |

- `candid.api_url` URL of the external authentication endpoint using Candid

  | Key          | candid.api_url    | Type: string | Scope: global |

- `candid.domains` Allowed Candid domains

  | Key          | candid.domains    | Type: string | Scope: global |

  Specify a comma-separated list of allowed Candid domains. An empty string means all domains are valid.

- `candid.expiry` Candid macaroon expiry

  | Key          | candid.expiry    | Type: integer | Default: 3600 | Scope: global |

  Specify the expiry time in seconds.

- `rbac.agent.private_key` Private key of the Candid agent

  | Key          | rbac.agent.private_key | Type: string | Scope: global |

  Specify the private key as provided during RBAC registration.

- `rbac.agent.public_key` Public key of the Candid agent

  | Key          | rbac.agent.public_key | Type: string | Scope: global |

  Specify the public key as provided during RBAC registration.

- `rbac.agent.url` URL of the Candid agent
Specify the URL as provided during RBAC registration.

\[ \text{rbac.agent.url} \]

\[ \text{rbac.agent.username} \] User name of the Candid agent

Specify the user name as provided during RBAC registration.

\[ \text{rbac.api.expiry} \] RBAC macaroon expiry

Specify the expiry time in seconds.

\[ \text{rbac.api.key} \] Public key of the RBAC server

\[ \text{rbac.api.url} \] URL of the external RBAC server

**OpenID Connect configuration**

The following server options configure external user authentication through OpenID Connect.

\[ \text{oidc.audience} \] Expected audience value for the application

This value is required by some providers.

\[ \text{oidc.client.id} \] OpenID Connect client ID
**OpenFGA configuration**

The following server options configure external user authorization through *Open Fine-Grained Authorization (Open-FGA)*: 

- `oidc.client.id` 
  - Key: `oidc.client.id` 
  - Type: `string` 
  - Scope: `global` 

- `oidc.issuer` OpenID Connect Discovery URL for the provider 
  - Key: `oidc.issuer` 
  - Type: `string` 
  - Scope: `global` 

- `openfga.api.token` API token of the OpenFGA server 
  - Key: `openfga.api.token` 
  - Type: `string` 
  - Scope: `global` 

- `openfga.api.url` URL of the OpenFGA server 
  - Key: `openfga.api.url` 
  - Type: `string` 
  - Scope: `global` 

- `openfga.store.id` ID of the OpenFGA permission store 
  - Key: `openfga.store.id` 
  - Type: `string` 
  - Scope: `global` 

- `openfga.store.model_id` ID of the OpenFGA authorization model 
  - Key: `openfga.store.model_id` 
  - Type: `string` 
  - Scope: `global`
Cluster configuration

The following server options control Clustering: cluster.healing_threshold Threshold when to evacuate an offline cluster member.

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster.healing_threshold</td>
<td>integer</td>
<td>0</td>
<td>global</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster.https_address</td>
<td>string</td>
<td></td>
<td>local</td>
</tr>
</tbody>
</table>

See Separate REST API and clustering networks.

cluster.images_minimal_replica Number of cluster members that replicate an image.

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster.images_minimal_replica</td>
<td>integer</td>
<td>3</td>
<td>global</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster.join_token_expiry</td>
<td>string</td>
<td>3H</td>
<td>global</td>
</tr>
</tbody>
</table>

cluster.max_standby Number of database stand-by members.

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster.max_standby</td>
<td>integer</td>
<td>2</td>
<td>global</td>
</tr>
</tbody>
</table>

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.
**Canonical LXD**

### cluster.max_voters

<table>
<thead>
<tr>
<th>Key</th>
<th>cluster.max_voters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>integer</td>
</tr>
<tr>
<td>Default</td>
<td>3</td>
</tr>
<tr>
<td>Scope</td>
<td>global</td>
</tr>
</tbody>
</table>

Specify the maximum number of cluster members that are assigned the database voter role. This must be an odd number \( \geq 3 \).

### cluster.offline_threshold

<table>
<thead>
<tr>
<th>Key</th>
<th>cluster.offline_threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>integer</td>
</tr>
<tr>
<td>Default</td>
<td>20</td>
</tr>
<tr>
<td>Scope</td>
<td>global</td>
</tr>
</tbody>
</table>

Specify the number of seconds after which an unresponsive member is considered offline.

**Images configuration**

The following server options configure how to handle images:

#### images.auto_update_cached

<table>
<thead>
<tr>
<th>Key</th>
<th>images.auto_update_cached</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>true</td>
</tr>
<tr>
<td>Scope</td>
<td>global</td>
</tr>
</tbody>
</table>

Whether to automatically update cached images

#### images.auto_update_interval

<table>
<thead>
<tr>
<th>Key</th>
<th>images.auto_update_interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>integer</td>
</tr>
<tr>
<td>Default</td>
<td>6</td>
</tr>
<tr>
<td>Scope</td>
<td>global</td>
</tr>
</tbody>
</table>

Interval at which to look for updates to cached images

Specify the interval in hours. To disable looking for updates to cached images, set this option to 0.

#### images.compression_algorithm

<table>
<thead>
<tr>
<th>Key</th>
<th>images.compression_algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>gzip</td>
</tr>
<tr>
<td>Scope</td>
<td>global</td>
</tr>
</tbody>
</table>

Compression algorithm to use for new images

Possible values are bzip2, gzip, lzma, xz, or none.

#### images.default_architecture

<table>
<thead>
<tr>
<th>Key</th>
<th>images.default_architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
</tbody>
</table>

Default architecture to use in a mixed-architecture cluster
images.remote_cache_expiry When an unused cached remote image is flushed

<table>
<thead>
<tr>
<th>Key:</th>
<th>images.remote_cache_expiry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>integer</td>
</tr>
<tr>
<td>Default:</td>
<td>10</td>
</tr>
<tr>
<td>Scope:</td>
<td>global</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Key:</th>
<th>loki.api.ca_cert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Scope:</td>
<td>global</td>
</tr>
</tbody>
</table>

`loki.api.url` URL to the Loki server

<table>
<thead>
<tr>
<th>Key:</th>
<th>loki.api.url</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Scope:</td>
<td>global</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Key:</th>
<th>loki.auth.password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Scope:</td>
<td>global</td>
</tr>
</tbody>
</table>

`loki.auth.username` User name used for Loki authentication

<table>
<thead>
<tr>
<th>Key:</th>
<th>loki.auth.username</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Scope:</td>
<td>global</td>
</tr>
</tbody>
</table>

`loki.labels` Labels for a Loki log entry

<table>
<thead>
<tr>
<th>Key:</th>
<th>loki.labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Scope:</td>
<td>global</td>
</tr>
</tbody>
</table>

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
### loki.types

Events to send to the Loki server

<table>
<thead>
<tr>
<th>Key:</th>
<th>loki.types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>lifecycle, logging</td>
</tr>
<tr>
<td>Scope:</td>
<td>global</td>
</tr>
</tbody>
</table>

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, MAAS integration, OVN integration, Backups and Storage:

- `backups.compression_algorithm` Compression algorithm to use for backups

<table>
<thead>
<tr>
<th>Key:</th>
<th>backups.compression_algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>gzip</td>
</tr>
<tr>
<td>Scope:</td>
<td>global</td>
</tr>
</tbody>
</table>

Possible values are `bzip2`, `gzip`, `lzma`, `xz`, or `none`.

- `instances.nic.host_name` How to set the host name for a NIC

<table>
<thead>
<tr>
<th>Key:</th>
<th>instances.nic.host_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>random</td>
</tr>
<tr>
<td>Scope:</td>
<td>global</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Key:</th>
<th>instances.placement.scriptlet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>string</td>
</tr>
<tr>
<td>Scope:</td>
<td>global</td>
</tr>
</tbody>
</table>

When using custom automatic instance placement logic, this option stores the scriptlet. See Instance placement scriptlet for more information.

- `maas.api.key` API key to manage MAAS
### Canonical LXD

<table>
<thead>
<tr>
<th>Key</th>
<th>maas.api.key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Scope</td>
<td>global</td>
</tr>
</tbody>
</table>

**maas.api.url** URL of the MAAS server

<table>
<thead>
<tr>
<th>Key</th>
<th>maas.api.url</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Scope</td>
<td>global</td>
</tr>
</tbody>
</table>

**maas.machine** Name of this LXD host in MAAS

<table>
<thead>
<tr>
<th>Key</th>
<th>maas.machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>host name</td>
</tr>
<tr>
<td>Scope</td>
<td>local</td>
</tr>
</tbody>
</table>

**network.ovn.integration_bridge** OVS integration bridge to use for OVN networks

<table>
<thead>
<tr>
<th>Key</th>
<th>network.ovn.integration_bridge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>br-int</td>
</tr>
<tr>
<td>Scope</td>
<td>global</td>
</tr>
</tbody>
</table>

**network.ovn.northbound_connection** OVN northbound database connection string

<table>
<thead>
<tr>
<th>Key</th>
<th>network.ovn.northbound_connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>unix:/var/run/ovn/ovnnb_db.sock</td>
</tr>
<tr>
<td>Scope</td>
<td>global</td>
</tr>
</tbody>
</table>

**storage.backups_volume** Volume to use to store backup tarballs

<table>
<thead>
<tr>
<th>Key</th>
<th>storage.backups_volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Scope</td>
<td>local</td>
</tr>
</tbody>
</table>

Specify the volume using the syntax **POOL/VOLUME**.

**storage.images_volume** Volume to use to store the image tarballs

<table>
<thead>
<tr>
<th>Key</th>
<th>storage.images_volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>string</td>
</tr>
<tr>
<td>Scope</td>
<td>local</td>
</tr>
</tbody>
</table>

Specify the volume using the syntax **POOL/VOLUME**.
2.4.17 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* 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 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 agroup quota. The btrfs.mount_options=compress-force storage pool option 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

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>btrfs.mount_options</td>
<td>string</td>
<td>user_subvol_rm_allowed</td>
<td>Mount options for block devices</td>
</tr>
<tr>
<td>size</td>
<td>string</td>
<td>auto (20% of free disk space, (\geq 5\ \text{GiB}) and (\leq 30\ \text{GiB}))</td>
<td>Size of the storage pool when creating loop-based pools (in bytes, suffixes supported, can be increased to grow storage pool)</td>
</tr>
<tr>
<td>source</td>
<td>string</td>
<td>-</td>
<td>Path to an existing block device, loop file or Btrfs subvolume</td>
</tr>
<tr>
<td>source.wipe</td>
<td>bool</td>
<td>false</td>
<td>Wipe the block device specified in source prior to creating the storage pool</td>
</tr>
</tbody>
</table>

Tip: In addition to these configurations, you can also set default values for the storage volume configurations. See *Configure default values for storage volumes.*
Storage volume configuration

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Condition</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>secured_shifted</td>
<td>bool</td>
<td>custom volume</td>
<td>same as volume.security.shifted or false</td>
<td>Enable ID shifting overlay (allows attach by multiple isolated instances)</td>
</tr>
<tr>
<td>secured_unmapped</td>
<td>bool</td>
<td>custom volume</td>
<td>same as volume.security.unmapped or false</td>
<td>Disable ID mapping for the volume</td>
</tr>
<tr>
<td>size</td>
<td>string</td>
<td>appropriate driver</td>
<td>same as volume.size</td>
<td>Size/quota of the storage volume</td>
</tr>
<tr>
<td>snapshots.expiry</td>
<td>string</td>
<td>custom volume</td>
<td>same as volume.snapshots.expiry</td>
<td>Controls when snapshots are to be deleted (expects an expression like 1M 2H 3d 4w 5m 6y)</td>
</tr>
<tr>
<td>snapshots.pattern</td>
<td>string</td>
<td>custom volume</td>
<td>same as volume.snapshots.pattern or snap%d</td>
<td>Pongo2 template string that represents the snapshot name (used for scheduled snapshots and unnamed snapshots)</td>
</tr>
<tr>
<td>snapshots.schedule</td>
<td>string</td>
<td>custom volume</td>
<td>same as volume.snapshots.schedule</td>
<td>Cron expression (&lt;minute&gt; &lt;hour&gt; &lt;dom&gt; &lt;month&gt; &lt;dow&gt;), a comma-separated list of schedule aliases (@hourly, @daily, @midnight, @weekly, @monthly, @annually, @yearly), or empty to disable automatic snapshots (the default)</td>
</tr>
</tbody>
</table>

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` server setting.

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Condition</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>string</td>
<td>appropriate driver</td>
<td>same as volume.size</td>
<td>Size/quota of the storage bucket</td>
</tr>
</tbody>
</table>
CephFS - cephfs

Ceph 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.

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.

cephpfs driver in LXD

Note: The cephfs driver can only be used for custom storage volumes with content type filesystem.

For other storage volumes, use the Ceph 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 must create the CephFS file system that you want to use beforehand and specify it through the source option.

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

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cephfs.cluster_name</td>
<td>string</td>
<td>ceph</td>
<td>Name of the Ceph cluster that contains the CephFS file system</td>
</tr>
<tr>
<td>cephfs.fscache</td>
<td>bool</td>
<td>false</td>
<td>Enable use of kernel fscache and cachefilesd</td>
</tr>
<tr>
<td>cephfs.path</td>
<td>string</td>
<td>/</td>
<td>The base path for the CephFS mount</td>
</tr>
<tr>
<td>cephfs.user.name</td>
<td>string</td>
<td>admin</td>
<td>The Ceph user to use</td>
</tr>
<tr>
<td>source</td>
<td>string</td>
<td>-</td>
<td>Existing CephFS file system or file system path to use</td>
</tr>
<tr>
<td>volatile.pool.pristine</td>
<td>string</td>
<td>true</td>
<td>Whether the CephFS file system was empty on creation time</td>
</tr>
</tbody>
</table>

Tip: In addition to these configurations, you can also set default values for the storage volume configurations. See Configure default values for storage volumes.
## Storage volume configuration

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Condition</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>security.type</td>
<td>bool</td>
<td>custom volume</td>
<td>same as volume.</td>
<td>Enable ID shifting overlay (allows attach by multiple isolated instances)</td>
</tr>
<tr>
<td>security.shift</td>
<td>bool</td>
<td>custom volume</td>
<td>security. shifted</td>
<td>Disable ID mapping for the volume</td>
</tr>
<tr>
<td>size</td>
<td>string</td>
<td>appropriate driver</td>
<td>same as volume.</td>
<td>Size/Quota of the storage volume</td>
</tr>
<tr>
<td>snapshots.expiry</td>
<td>string</td>
<td>custom volume</td>
<td>same as</td>
<td>Controls when snapshots are to be deleted (expects an expression like</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>snapshots.</td>
<td>1M 2H 3d 4w 5m 6y)</td>
</tr>
<tr>
<td>snapshots.pattern</td>
<td>string</td>
<td>custom volume</td>
<td>same as</td>
<td>Pongo2 template string that represents the snapshot name (used for</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>snapshots.</td>
<td>scheduled snapshots and unnamed snapshots)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>pattern or</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>snap%d</td>
<td></td>
</tr>
<tr>
<td>snapshots.schedule</td>
<td>string</td>
<td>custom volume</td>
<td>same as</td>
<td>Cron expression (&lt;minute&gt; &lt;hour&gt; &lt;dom&gt; &lt;month&gt; &lt;dow&gt;), a comma-separated</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>snapshots.</td>
<td>list of schedule aliases (@hourly, @daily,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>schedule</td>
<td>@midnight, @weekly, @monthly, @annually, @yearly), or empty to</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>disable automatic snapshots (the default)</td>
</tr>
</tbody>
</table>

1 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.
**Ceph Object - cephobject**

Ceph 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.

Ceph provides different components for block storage and for file systems.

Ceph Object Gateway is an object storage interface built on top of librados to provide applications with a RESTful gateway to Ceph Storage Clusters. 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 or CephFS 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 for information on how to set up a Ceph cluster and Ceph Object Gateway on how to set up a radosgw environment.

The radosgw URL can be specified at pool creation time using the cephobject.radosgw.endpoint option.

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.
Canonical LXD

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

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cephobject.bucket.name_prefix</td>
<td>string</td>
<td>-</td>
<td>Prefix to add to bucket names in Ceph</td>
</tr>
<tr>
<td>cephobject.cluster_name</td>
<td>string</td>
<td>ceph</td>
<td>The Ceph cluster to use</td>
</tr>
<tr>
<td>cephobject.radosgw.endpoint</td>
<td>string</td>
<td>-</td>
<td>URL of the radosgw gateway process</td>
</tr>
<tr>
<td>cephobject.radosgw.endpoint_cert_file</td>
<td>string</td>
<td>-</td>
<td>Path to the file containing the TLS client certificate to use for endpoint communication</td>
</tr>
<tr>
<td>cephobject.user.name</td>
<td>string</td>
<td>admin</td>
<td>The Ceph user to use</td>
</tr>
<tr>
<td>volatile.pool.pristine</td>
<td>string</td>
<td>true</td>
<td>Whether the radosgw lxd-admin user existed at creation time</td>
</tr>
</tbody>
</table>

Storage bucket configuration

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>string</td>
<td>-</td>
<td>Quota of the storage bucket</td>
</tr>
</tbody>
</table>

Ceph RBD - ceph

Ceph 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.

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.
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`).

Alternatively, you can use the *CephFS* 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 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* 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` configuration option to the erasure coded pool name and the `source` 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

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ceph.cluster_name</td>
<td>string</td>
<td>ceph</td>
<td>Name of the Ceph cluster in which to create new storage pools</td>
</tr>
<tr>
<td>ceph.osd.data_pool_name</td>
<td>string</td>
<td>-</td>
<td>Name of the OSD data pool</td>
</tr>
<tr>
<td>ceph.osd.pg_num</td>
<td>string</td>
<td>32</td>
<td>Number of placement groups for the OSD storage pool</td>
</tr>
<tr>
<td>ceph.osd.pool_name</td>
<td>string</td>
<td>name of the pool</td>
<td>Name of the OSD storage pool</td>
</tr>
<tr>
<td>ceph.rbd.clone_copy</td>
<td>bool</td>
<td>true</td>
<td>Whether to use RBD lightweight clones rather than full dataset copies</td>
</tr>
<tr>
<td>ceph.rbd.du</td>
<td>bool</td>
<td>true</td>
<td>Whether to use RBD du to obtain disk usage data for stopped instances</td>
</tr>
<tr>
<td>ceph.rbd.features</td>
<td>string</td>
<td>layering</td>
<td>Comma-separated list of RBD features to enable on the volumes</td>
</tr>
<tr>
<td>ceph.user.name</td>
<td>string</td>
<td>admin</td>
<td>The Ceph user to use when creating storage pools and volumes</td>
</tr>
<tr>
<td>source</td>
<td>string</td>
<td>-</td>
<td>Existing OSD storage pool to use</td>
</tr>
<tr>
<td>volatile.pool.pristine</td>
<td>string</td>
<td>true</td>
<td>Whether the pool was empty on creation time</td>
</tr>
</tbody>
</table>

Tip: In addition to these configurations, you can also set default values for the storage volume configurations. See Configure default values for storage volumes.
# Storage volume configuration

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Condition</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>block.</td>
<td>string</td>
<td>block-based volume with content type filesystem</td>
<td>same as volume. block. filesystem</td>
<td>File system of the storage volume: btrfs, ext4 or xfs (ext4 if not set)</td>
</tr>
<tr>
<td>mount.</td>
<td>string</td>
<td>block-based volume with content type filesystem</td>
<td>same as volume. block. mount_options</td>
<td>Mount options for block-backed file system volumes</td>
</tr>
<tr>
<td>shifted</td>
<td>bool</td>
<td>custom volume</td>
<td>same as volume. security. shifted or false</td>
<td>Enable ID shifting overlay (allows attach by multiple isolated instances)</td>
</tr>
<tr>
<td>unmapped</td>
<td>bool</td>
<td>custom volume</td>
<td>same as volume. security. unmapped or false</td>
<td>Disable ID mapping for the volume</td>
</tr>
<tr>
<td>size</td>
<td>string</td>
<td>custom volume</td>
<td>same as volume. size</td>
<td>Size/quota of the storage volume</td>
</tr>
<tr>
<td>expiry</td>
<td>string</td>
<td>custom volume</td>
<td>same as volume. snapshots. expiry</td>
<td>Controls when snapshots are to be deleted (expects an expression like 1M 2H 3d 4w 5m 6y)</td>
</tr>
<tr>
<td>pattern</td>
<td>string</td>
<td>custom volume</td>
<td>same as volume. snapshots. pattern</td>
<td>Pongo2 template string that represents the snapshot name (used for scheduled snapshots and unnamed snapshots)</td>
</tr>
<tr>
<td>schedule</td>
<td>string</td>
<td>custom volume</td>
<td>same as volume. snapshots. schedule</td>
<td>Cron expression (&lt;minute&gt; &lt;hour&gt; &lt;dom&gt; &lt;month&gt; &lt;dow&gt;), a comma-separated list of schedule aliases (@hourly, @daily, @midnight, @weekly, @monthly, @annually, @yearly), or empty to disable automatic snapshots (the default)</td>
</tr>
</tbody>
</table>

---

1 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.
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 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

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsync.bwlimit</td>
<td>string</td>
<td>0 (no limit)</td>
<td>The upper limit to be placed on the socket I/O when rsync must be used to transfer storage entities</td>
</tr>
<tr>
<td>rsync.compression</td>
<td>bool</td>
<td>true</td>
<td>Whether to use compression while migrating storage pools</td>
</tr>
<tr>
<td>source</td>
<td>string</td>
<td>-</td>
<td>Path to an existing directory</td>
</tr>
</tbody>
</table>

Tip: In addition to these configurations, you can also set default values for the storage volume configurations. See Configure default values for storage volumes.
## Storage volume configuration

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Condition</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>security.shifted</td>
<td>bool</td>
<td>custom volume</td>
<td>same as volume. security. shifted or false</td>
<td>Enable ID shifting overlay (allows attach by multiple isolated instances)</td>
</tr>
<tr>
<td>security.unmapped</td>
<td>bool</td>
<td>custom volume</td>
<td>same as volume. security. unmapped or false</td>
<td>Disable ID mapping for the volume</td>
</tr>
<tr>
<td>size</td>
<td>string</td>
<td>appropriate driver</td>
<td>same as volume.size</td>
<td>Size/quota of the storage volume</td>
</tr>
<tr>
<td>snapshots.expiry</td>
<td>string</td>
<td>custom volume</td>
<td>same as volume. snapshots. expiry</td>
<td>Controls when snapshots are to be deleted (expects an expression like 1M 2H 3d 4w 5m 6y)</td>
</tr>
<tr>
<td>snapshots.pattern</td>
<td>string</td>
<td>custom volume</td>
<td>same as volume. snapshots. pattern or snap%d</td>
<td>Pongo2 template string that represents the snapshot name (used for scheduled snapshots and unnamed snapshots)</td>
</tr>
<tr>
<td>snapshots.schedule</td>
<td>string</td>
<td>custom volume</td>
<td>same as volume. snapshots. schedule</td>
<td>Cron expression (&lt;minute&gt; &lt;hour&gt; &lt;dom&gt; &lt;month&gt; &lt;dow&gt;), a comma-separated list of schedule aliases (@hourly, @daily, @midnight, @weekly, @monthly, @annually, @yearly), or empty to disable automatic snapshots (the default)</td>
</tr>
</tbody>
</table>

## 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` 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.

---

1. 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.
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 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 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 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 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**

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lvm.thinpool_name</td>
<td>string</td>
<td>LXDThinPool</td>
<td>Thin pool where volumes are created</td>
</tr>
<tr>
<td>lvm.thinpool_metadata</td>
<td>string</td>
<td>0 (auto)</td>
<td>The size of the thin pool metadata volume (the default is to let LVM calculate an appropriate size)</td>
</tr>
<tr>
<td>lvm.use_thinpool</td>
<td>bool</td>
<td>true</td>
<td>Whether the storage pool uses a thin pool for logical volumes</td>
</tr>
<tr>
<td>lvm.vg.force_reuse</td>
<td>bool</td>
<td>false</td>
<td>Force using an existing non-empty volume group</td>
</tr>
<tr>
<td>lvm.vg_name</td>
<td>string</td>
<td>name of the pool</td>
<td>Name of the volume group to create</td>
</tr>
<tr>
<td>rsync.bwlimit</td>
<td>string</td>
<td>0 (no limit)</td>
<td>The upper limit to be placed on the socket I/O when rsync must be used to transfer storage entities</td>
</tr>
<tr>
<td>rsync.compression</td>
<td>bool</td>
<td>true</td>
<td>Whether to use compression while migrating storage pools</td>
</tr>
<tr>
<td>size</td>
<td>string</td>
<td>auto (20% of free disk space, &gt;= 5 GiB and &lt;= 30 GiB)</td>
<td>Size of the storage pool when creating loop-based pools (in bytes, suffixes supported, can be increased to grow storage pool)</td>
</tr>
<tr>
<td>source</td>
<td>string</td>
<td>-</td>
<td>Path to an existing block device, loop file or LVM volume group</td>
</tr>
<tr>
<td>source.wipe</td>
<td>bool</td>
<td>false</td>
<td>Wipe the block device specified in source prior to creating the storage pool</td>
</tr>
</tbody>
</table>

**Tip:** In addition to these configurations, you can also set default values for the storage volume configurations. See [Configure default values for storage volumes](#).
## Storage volume configuration

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Condition</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>block. filesystem</td>
<td>string</td>
<td>block-based volume with content type filesystem</td>
<td>same as volume. block. filesystem</td>
<td>File system of the storage volume: btrfs, ext4 or xfs (ext4 if not set)</td>
</tr>
<tr>
<td>block. mount_options</td>
<td>string</td>
<td>block-based volume with content type filesystem</td>
<td>same as volume. block. mount_options</td>
<td>Mount options for block-backed file system volumes</td>
</tr>
<tr>
<td>lvm. stripe : size</td>
<td>string</td>
<td>same as volume. lvm. stripes</td>
<td>Size of stripes to use (at least 4096 bytes and multiple of 512 bytes)</td>
<td></td>
</tr>
<tr>
<td>security. shifted</td>
<td>bool</td>
<td>custom volume</td>
<td>same as volume. security. shifted or false</td>
<td>Enable ID shifting overlay (allows attach by multiple isolated instances)</td>
</tr>
<tr>
<td>security. unmapped</td>
<td>bool</td>
<td>custom volume</td>
<td>same as volume. security. unmapped or false</td>
<td>Disable ID mapping for the volume</td>
</tr>
<tr>
<td>size</td>
<td>string</td>
<td>same as volume. size</td>
<td>Size/quota of the storage volume</td>
<td></td>
</tr>
<tr>
<td>snapshot. expiry</td>
<td>string</td>
<td>custom volume</td>
<td>same as volume. snapshots. expiry</td>
<td>Controls when snapshots are to be deleted (expects an expression like 1M 2H 3d 4w 5m 6y)</td>
</tr>
</tbody>
</table>
| snapshot. pattern | string          | custom volume | same as volume. snapshots. pattern or snap%d | Pongo2 template string that represents the snapshot name (used for scheduled snapshots and unnamed snapshots)

1 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
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` server setting.

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Condition</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>string</td>
<td>appropriate driver</td>
<td>same as <code>volume.size</code></td>
<td>Size/quota of the storage bucket</td>
</tr>
</tbody>
</table>

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/ then incremented by one for the new name.
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 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 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` 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` 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 quota for your storage volume. If you want to use the `refquota` property instead, set the `zfs.use_refquota` 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.use_reserve_space` (or `volume.zfs.use_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**

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>string</td>
<td>auto (20% of free disk space, (&gt;= 5 \text{ GiB} ) and (&lt;= 30 \text{ GiB} ))</td>
<td>Size of the storage pool when creating loop-based pools (in bytes, suffixes supported, can be increased to grow storage pool)</td>
</tr>
<tr>
<td>source</td>
<td>string</td>
<td>-</td>
<td>Path to an existing block device, loop file or ZFS dataset/pool</td>
</tr>
<tr>
<td>source.</td>
<td>bool</td>
<td>false</td>
<td>Wipe the block device specified in <code>source</code> prior to creating the <code>zfs</code> storage pool</td>
</tr>
<tr>
<td>zfs. clone_coq</td>
<td>string</td>
<td>true</td>
<td>Whether to use ZFS lightweight clones rather than full copies (Boolean), or <code>rebase</code> to copy based on the initial image</td>
</tr>
<tr>
<td>zfs. export</td>
<td>bool</td>
<td>true</td>
<td>Disable zpool export while unmount performed</td>
</tr>
<tr>
<td>zfs. pool_name</td>
<td>string</td>
<td>name of the pool</td>
<td>Name of the zpool</td>
</tr>
</tbody>
</table>

**Tip:** In addition to these configurations, you can also set default values for the storage volume configurations. See [Configure default values for storage volumes](#).
# Storage volume configuration

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Condition</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>block.</td>
<td>string</td>
<td>block-based volume</td>
<td>same as volume.</td>
<td>File system of the storage volume: btrfs, ext4 or xfs (ext4 if not set)</td>
</tr>
<tr>
<td>filesystem</td>
<td></td>
<td>with content type filesystem (zfs.block_mode enabled)</td>
<td>filesystem</td>
<td></td>
</tr>
<tr>
<td>block.</td>
<td>string</td>
<td>block-based volume</td>
<td>same as volume.</td>
<td>Mount options for block-backed file system volumes</td>
</tr>
<tr>
<td>mount.</td>
<td></td>
<td>with content type filesystem (zfs.block_mode enabled)</td>
<td>mount_options</td>
<td></td>
</tr>
<tr>
<td>shifted</td>
<td>bool</td>
<td>custom volume</td>
<td>same as volume.</td>
<td>Enable ID shifting overlay (allows attach by multiple isolated instances)</td>
</tr>
<tr>
<td>security.</td>
<td></td>
<td></td>
<td>shifted</td>
<td></td>
</tr>
<tr>
<td>unmapped</td>
<td>bool</td>
<td>custom volume</td>
<td>same as volume.</td>
<td>Disable ID mapping for the volume</td>
</tr>
<tr>
<td>security.</td>
<td></td>
<td></td>
<td>unmapped or false</td>
<td></td>
</tr>
<tr>
<td>size</td>
<td>string</td>
<td></td>
<td>same as volume.</td>
<td>Size/quota of the storage volume</td>
</tr>
<tr>
<td>snapshots.</td>
<td>expiry</td>
<td>custom volume</td>
<td>same as volume.</td>
<td>Controls when snapshots are to be deleted (expects an expression like 1M 2H 3d 4w 5m 6y)</td>
</tr>
<tr>
<td>snapshots.</td>
<td>pattern</td>
<td>custom volume</td>
<td>same as volume.</td>
<td>Pongo2 template string that represents the snapshot name (used for scheduled snapshots and unnamed snapshots)</td>
</tr>
<tr>
<td>schedule</td>
<td>string</td>
<td>custom volume</td>
<td>same as snapshots.</td>
<td>Cron expression (&lt;minute&gt; &lt;hour&gt; &lt;dom&gt; &lt;month&gt; &lt;dow&gt;), a comma-separated list of schedule aliases (@hourly, @daily, @midnight, @weekly, @monthly, @annually, @yearly), or empty to disable automatic snapshots (the default)</td>
</tr>
<tr>
<td>zfs.</td>
<td>blocks</td>
<td></td>
<td>same as volume.</td>
<td>Size of the ZFS block in range from 512 to 16 MiB (must be power of 2) - for block volume, a maximum value of 128 KiB will be used even if a higher value is set</td>
</tr>
<tr>
<td>zfs.</td>
<td>block</td>
<td></td>
<td>same as volume.</td>
<td>Whether to use a formatted zvol rather than a (zfs. block_mode can be set only for custom storage volumes; use volume.zfs.block_mode to enable ZFS block mode for all storage volumes in the pool, including instance volumes)</td>
</tr>
<tr>
<td>zfs.</td>
<td>delegate</td>
<td>ZFS 2.2 or higher</td>
<td>same as volume.</td>
<td>Controls whether to delegate the ZFS dataset and anything underneath it to the container(s) using it. Allows the use of the zfs command in the container.</td>
</tr>
<tr>
<td>zfs.</td>
<td>bool</td>
<td></td>
<td>same as volume.</td>
<td>Remove snapshots as needed</td>
</tr>
</tbody>
</table>
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` server setting.

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Condition</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>string</td>
<td>appropriate driver</td>
<td>same as volume.size</td>
<td>Size/quota of the storage bucket</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Directory</th>
<th>Btrfs</th>
<th>LVM</th>
<th>ZFS</th>
<th>Ceph RBD</th>
<th>CephFS</th>
<th>Ceph Object</th>
<th>Ob-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimized image storage</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>n/a</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>Optimized instance creation</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>n/a</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>Optimized snapshot creation</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>Optimized image transfer</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>n/a</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>Optimized volume transfer</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>Copy on write</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>Block based</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>Instant cloning</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>Storage driver usable inside a container</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>n/a</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>Restore from older snapshots (not latest)</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>Storage quotas</td>
<td>yes²</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Available on lxd init</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>Object storage</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td></td>
</tr>
</tbody>
</table>

¹ The `snapshots.pattern` option takes a Pongo2 template string to format the snapshot name.
² To add a timestamp 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.

² Requires `zfs.delegate` to be enabled.

The `dir` driver supports storage quotas when running on either ext4 or XFS with project quotas enabled at the file system level.
Optimized image storage

All storage drivers except for the directory driver 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.

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. However, the full potential of the optimized transfer becomes apparent when refreshing a copy of an instance or custom volume that uses periodic snapshots. With optimized transfer, 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. 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 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 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, \texttt{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

Explanation:

- About storage pools, volumes and buckets
CONFIGURATION OPTIONS

cluster
 scheduler.instance, 201
 user.*, 201

instance
 agent.nic_config, 237
 boot.autostart, 239
 boot.autostart.delay, 239
 boot.autostart.priority, 239
 boot.host_shutdown_timeout, 239
 boot.stop.priority, 239
 cloud-init.network-config, 240
 cloud-init.user-data, 240
 cloud-init.vendor-data, 240
 cluster.evacuate, 237
 environment.*, 238
 limits.cpu, 241
 limits.cpu.allowance, 241
 limits.cpu.nodes, 241
 limits.cpu.priority, 241
 limits.disk.priority, 242
 limits.hugepages.1GB, 242
 limits.hugepages.1MB, 242
 limits.hugepages.2MB, 242
 limits.hugepages.64KB, 243
 limits.kernel.*, 244
 limits.memory, 243
 limits.memory.enforce, 243
 limits.memory.hugepages, 243
 limits.memory.swap, 243
 limits.memory.swap.priority, 244
 limits.network.priority, 244
 limits.processes, 244
 linux.kernel_modules, 238
 linux.sysctl.*, 238
 migration.incremental.memory, 247
 migration.incremental.memory.goal, 247
 migration.incremental.memory.iterations, 248
 migration.stateful, 248
 nvidia.driver.capabilities, 248
 nvidia.require.cuda, 248
 nvidia.require.driver, 248
 nvidia.runtime, 249
 raw.apparmor, 249
 raw.idmap, 249
 raw.lxc, 249
 raw.qemu, 249
 raw.qemu.conf, 249
 raw.seccomp, 250
 security.agent.metrics, 251
 security.csm, 251
 security.devlxd, 252
 security.devlxd.images, 252
 security.idmap.base, 252
 security.idmap.isolated, 252
 security.idmap.size, 252
 security.nesting, 253
 security.privileged, 253
 security.protection.delete, 253
 security.protection.shift, 253
 security.secureboot, 253
 security.sev, 253
 security.sev.policy.es, 254
 security.sev.session.data, 254
 security.sev.session.dh, 254
 security.syscalls.allow, 254
 security.syscalls.deny, 254
 security.syscalls.deny_compat, 254
 security.syscalls.deny_default, 255
 security.syscalls.intercept.bpf, 255
 security.syscalls.intercept.bpf.devices, 255
 security.syscalls.intercept.mkmod, 255
 security.syscalls.intercept.mount, 255
 security.syscalls.intercept.mount.allowed, 255
 security.syscalls.intercept.mount.fuse, 256
 security.syscalls.intercept.mount.shift, 256
 security.syscalls.intercept.sched_setscheduler, 256
 security.syscalls.intercept.setxattr, 256
 security.syscalls.intercept.sysinfo, 256
 snapshots.expiry, 257
 snapshots.pattern, 257
 snapshots.schedule, 257
snapshots.schedule.stopped, 257
user.*, 238
user.network-config, 240
user.user-data, 240
user.vendor-data, 240
volatile.<name>.apply_quota, 258
volatile.<name>.ceph_rbd, 258
volatile.<name>.host_name, 258
volatile.<name>.hwaddr, 258
volatile.<name>.last_state.created, 258
volatile.<name>.last_state.hwaddr, 258
volatile.<name>.last_state.ip_addresses, 259
volatile.<name>.last_state.mtu, 259
volatile.<name>.last_state.vdpa.name, 259
volatile.<name>.last_state.vf.hwaddr, 259
volatile.<name>.last_state.vf.id, 259
volatile.<name>.last_state.vf.spoofcheck, 259
volatile.<name>.last_state.vf.vlan, 259
volatile.apply_nvram, 260
volatile.apply_template, 260
volatile.base_image, 260
volatile.cloud_init.instance-id, 260
volatile.evacuate.origin, 260
volatile.idmap.base, 260
volatile.idmap.current, 260
volatile.idmap.next, 260
volatile.last_state.idmap, 261
volatile.last_state.power, 261
volatile.uuid, 261
volatile.uuid.generation, 261
volatile.vssock_id, 261
project
backups.compression_algorithm, 514
features.images, 506
features.networks, 506
features.networks.zones, 506
features.profiles, 506
features.storage.buckets, 507
features.storage.volumes, 507
images.auto_update Cached, 514
images.auto_update_interval, 514
images.compression_algorithm, 514
images.default_architecture, 514
images.remote_cache_expiry, 514
limits.containers, 507
limits.cpu, 507
limits.disk, 508
limits.instances, 508
limits.memory, 508
limits.networks, 508
limits.processes, 508
limits.virtual-machines, 508
restricted, 509
restricted.backups, 509
restricted.cluster.groups, 509
restricted.cluster.target, 509
restricted.containers.interception, 510
restricted.containers.lowlevel, 510
restricted.containers.nesting, 510
restricted.containers.privilege, 510
restricted.devices.disk, 510
restricted.devices.disk.paths, 511
restricted.devices.gpu, 511
restricted.devices.infiniband, 511
restricted.devices.nic, 511
restricted.devices.pci, 511
restricted.devices.proxy, 511
restricted.devices.unix-block, 512
restricted.devices.unix-char, 512
restricted.devices.unix-hotplug, 512
restricted.devices.usb, 512
restricted.idmap.gid, 512
restricted.idmap.uid, 512
restricted.networks.access, 513
restricted.networks.subnets, 513
restricted.networks.uplinks, 513
restricted.networks.zones, 513
restricted.snapshots, 513
restricted.virtual-machines.lowlevel, 513
user.*, 514
server
acme.agree_tos, 590
acme.ca_url, 590
acme.domain, 590
acme.email, 590
backups.compression_algorithm, 597
candid.api_key, 591
candid.api_url, 591
candid.domains, 591
candid.expiry, 591
cluster.healing_threshold, 594
cluster.https_address, 594
cluster.images_minimal_replica, 594
cluster.join_token_expiry, 594
cluster.max_standby, 594
cluster.max_voters, 594
cluster.offline_threshold, 595
core.bgp_address, 587
core.bgp_asn, 587
core.bgp_routerid, 587
core.debug_address, 587
core.dns_address, 587
core.https_address, 587
core.https_allowed_credentials, 587
core.https_allowed_headers, 588
core.https_allowed_methods, 588
core.https_allowed_origin, 588
core.https_trusted_proxy, 588
core.metrics_address, 588
core.metrics_authentication, 588
core.proxy_http, 588
core.proxy_https, 589
core.proxy_ignore_hosts, 589
core.remote_token_expiry, 589
core.shutdown_timeout, 589
core.storage_buckets_address, 589
core.syslog_socket, 589
core.trust_ca_certificates, 590
core.trust_password, 590
images.auto_update_cached, 595
images.auto_update_interval, 595
images.compression_algorithm, 595
images.default_architecture, 595
images.remote_cache_expiry, 596
instances.nic.host_name, 597
instances.placement.scriptlet, 597
loki.api.ca_cert, 596
loki.api.url, 596
loki.auth.password, 596
loki.auth.username, 596
loki.labels, 596
loki.loglevel, 596
loki.types, 597
maas.api.key, 597
maas.api.url, 598
maas.machine, 598
network.ovn.integration_bridge, 598
network.ovn.northbound_connection, 598
oidc.audience, 592
oidc.client.id, 592
oidc.issuer, 593
openfga.api.token, 593
openfga.api.url, 593
openfga.store.id, 593
openfga.store.model_id, 593
rbac.agent.private_key, 591
rbac.agent.public_key, 591
rbac.agent.url, 591
rbac.agent.username, 592
rbac.api.expiry, 592
rbac.api.key, 592
rbac.api.url, 592
storage.backups_volume, 598
storage.images_volume, 598