Canonical Kubernetes

Canonical Group Ltd

May 10, 2024

CONTENTS

Canonical Kubernetes is a performant, lightweight, secure and opinionated distribution of **Kubernetes** which includes everything needed to create and manage a scalable cluster suitable for all use cases.

You can find out more about Canonical Kubernetes on this *overview page* or see a more detailed explanation in our *architecture documentation*.

CHAPTER

ONE

CANONICAL KUBERNETES SNAP DOCUMENTATION

The Canonical Kubernetes snap is a performant, lightweight, secure and opinionated distribution of **Kubernetes** which includes everything needed to create and manage a scalable cluster suitable for all use cases.

You can find out more about Canonical Kubernetes on this *overview page* or see a more detailed explanation in our *architecture documentation*.

For deployment at scale, Canonical Kubernetes is also available as a *Juju charm*

1.1 In this documentation

Tutorial **Start here!** A hands-on introduction to Canonical K8s 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

1.2 Project and community

Canonical Kubernetes is a member of the Ubuntu family. It's an open source project which welcomes community involvement, contributions, suggestions, fixes and constructive feedback.

- Our Code of Conduct
- Our *community*
- How to *contribute*
- Our development *roadmap*

CHAPTER

TWO

TUTORIALS

This section contains a step-by-step guide to help you start exploring how to install and use Canonical Kubernetes.

2.1 Getting started

Installing Canonical Kubernetes should only take a few minutes. This tutorial explains how to install the snap package and some typical operations.

2.1.1 What you will need

- An Ubuntu 22.04 LTS or 20.04 LTS environment to run the commands (or another operating system which supports snapd - see the snapd documentation)
- System Requirements: Your machine should have at least 40G disk space and 4G of memory

1. Install Canonical Kubernetes

Install the Canonical Kubernetes snap with:

```
sudo snap install k8s --edge --classic
```

2. Bootstrap a Kubernetes Cluster

Bootstrap a Kubernetes cluster with default configuration using:

```
sudo k8s bootstrap
```

This command initialises your cluster and configures your host system as a Kubernetes node. For custom configurations, you can explore additional options using:

```
sudo k8s bootstrap --help
```

3. Check cluster status

To confirm the installation was successful and your node is ready you should run:

```
sudo k8s status
```

Run the following command to list all the pods in the `kube-system` namespace:

```
sudo k8s kubectl get pods -n kube-system
```

You will observe at least three pods running:

- **CoreDNS**: Provides DNS resolution services.
- **Network operator**: Manages the life-cycle of the networking solution.
- **Network agent**: Facilitates network management.

Confirm that Canonical Kubernetes has transitioned to the `k8s is ready` state by running:

```
sudo k8s status --wait-ready
```

5. Access Kubernetes

The standard tool for deploying and managing workloads on Kubernetes is kubectl. For convenience, Canonical Kubernetes bundles a version of kubectl for you to use with no extra setup or configuration. For example, to view your node you can run the command:

```
sudo k8s kubectl get nodes
```

…or to see the running services:

```
sudo k8s kubectl get services
```

6. Deploy an app

Kubernetes is meant for deploying apps and services. You can use the `kubectl` command to do that as with any Kubernetes.

Let's deploy a demo NGINX server:

```
sudo k8s kubectl create deployment nginx --image=nginx
```

This command launches a pod, the smallest deployable unit in Kubernetes, running the NGINX application within a container.

You can check the status of your pods by running:

```
sudo k8s kubectl get pods
```

This command shows all pods in the default namespace. It may take a moment for the pod to be ready and running.

7. Remove an app

To remove the NGINX workload, execute the following command:

```
sudo k8s kubectl delete deployment nginx
```

To verify that the pod has been removed, you can check the status of pods by running:

```
sudo k8s kubectl get pods
```

8. Enable Local Storage

In scenarios where you need to preserve application data beyond the life-cycle of the pod, Kubernetes provides persistent volumes.

With Canonical Kubernetes, you can enable local-storage to configure your storage solutions:

```
sudo k8s enable local-storage
```

To verify that the local-storage is enabled, execute:

```
sudo k8s status
```

You should see `local-storage enabled` in the command output.

Let's create a `PersistentVolumeClaim` and use it in a `Pod`. For example, we can deploy the following manifest:

```
sudo k8s kubectl apply -f https://raw.githubusercontent.com/canonical/k8s-snap/main/docs/
↪src/assets/tutorial-pod-with-pvc.yaml
```

This command deploys a pod based on the YAML configuration of a storage writer pod and a persistent volume claim with a capacity of 1G.

To confirm that the persistent volume is up and running:

```
sudo k8s kubectl get pvc myclaim
```

You can inspect the storage-writer-pod with:

```
sudo k8s kubectl describe pod storage-writer-pod
```

9. Disable Local Storage

Begin by removing the pod along with the persistent volume claim:

```
sudo k8s kubectl delete pvc myclaim
sudo k8s kubectl delete pod storage-writer-pod
```

Next, disable the local storage:

```
sudo k8s disable local-storage
```

10. Remove Canonical Kubernetes (Optional)

To uninstall the Canonical Kubernetes snap, execute:

```
sudo snap remove k8s
```

This command removes the `k8s` snap and automatically creates a snapshot of all data for future restoration.

If you wish to remove the snap without saving a snapshot of its data, add `--purge` to the command:

```
sudo snap remove k8s --purge
```

This option ensures complete removal of the snap and its associated data.

2.1.2 Next Steps

- Keep mastering Canonical Kubernetes with kubectl: *How to use kubectl*
- Explore Kubernetes commands with our *Command Reference Guide*
- Learn how to set up a multi-node environment *Setting up a K8s cluster*
- Configure storage options *Storage*
- Master Kubernetes networking concepts: *Networking*
- Discover how to enable and configure Ingress resources *Ingress*

2.2 Basic operations with Kubernetes using kubectl

Kubernetes provides a command line tool for communicating with a Kubernetes cluster's control plane, using the Kubernetes API. This guide outlines how some of the everyday operations of your Kubernetes cluster can be managed with this tool.

2.2.1 What you will need

Before you begin, make sure you have the following:

- A bootstrapped Canonical Kubernetes cluster (See *Getting Started*)
- You are using the built-in `kubectl` command from the snap.

1. The Kubectl Command

The `kubectl` command communicates with the Kubernetes API server.

The `kubectl` command included with Canonical Kubernetes is built from the original upstream source into the `k8s` snap you have installed.

2. How To Use Kubectl

To access `kubectl`, run the following command:

```
sudo k8s kubectl <command>
```

> **Note**: Only control plane nodes can use the `kubectl` command. Worker nodes do not have access to this command.

3. Configuration

In Canonical Kubernetes, the `kubeconfig` file that is being read to display the configuration when you run `kubectl config view` lives at `/etc/kubernetes/admin.conf`. You can change this by setting a `KUBECONFIG` environment variable or passing the `--kubeconfig` flag to a command.

To find out more, you can visit the official kubeconfig documentation

4. Viewing objects

Let's review what was created in the *Getting Started* guide.

To see what pods were created when we enabled the `network` and `dns` components:

```
sudo k8s kubectl get pods -o wide -n kube-system
```

You should be seeing the network operator, networking agent and CoreDNS pods.

> **Note**: If you see an error message here, it is likely that you forgot to bootstrap your cluster.

```
sudo k8s kubectl get services --all-namespace
```

The `kubernetes` service in the `default` namespace is where the Kubernetes API server resides, and it's the endpoint with which other nodes in your cluster will communicate.

5. Creating and Managing Objects

Let's deploy an NGINX server using this command:

```
sudo k8s kubectl create deployment nginx --image=nginx:latest
```

To observe the NGINX pod running in the default namespace:

```
sudo k8s kubectl get pods
```

Let's now scale this deployment, which means increasing the number of pods it manages.

```
sudo k8s kubectl scale deployment nginx --replicas=3
```

Execute `sudo k8s kubectl get pods` again and notice that you have 3 NGINX pods.

Let's delete those 3 pods to demonstrate a deployment's ability to ensure the declared state of the cluster is maintained.

First, open a new terminal so you can watch the changes as they happen. Run this command in a new terminal:

```
sudo k8s kubectl get pods --all-namespace --watch
```

Now, go back to your original terminal and run:

```
sudo k8s kubectl delete pods -l app=nginx
```

The above command deletes all pods in the cluster that are labelled with `app=nginx`.

You'll notice the original 3 pods will have a status of `Terminating` and 3 new pods will have a status of `ContainerCreating`.

2.2.2 Further information

- Explore Kubernetes commands with our *Command Reference Guide*
- See the official `kubectl` reference https://kubernetes.io/docs/reference/kubectl/

2.3 Adding and Removing Nodes

Typical production clusters are hosted across multiple data centres and cloud environments, enabling them to leverage geographical distribution for improved availability and resilience.

This tutorial simplifies the concept by creating a cluster within a controlled environment using two Multipass VMs. The approach here allows us to focus on the foundational aspects of clustering using Canonical Kubernetes without the complexities of a full-scale, production setup. If your nodes are already installed, you can skip the multipass setup and go to *step 2*.

2.3.1 Before starting

In this article, "**control plane**" refers to the Multipass VM that operates the control plane, while "**worker**" denotes the Multipass VM running the worker node.

2.3.2 What you will need

- Multipass (See Multipass Installation)

1. Create both VMs

The first step is creating the VMs.

```
multipass launch 22.04 --name control-plane -m 4G -d 8G
```

```
multipass launch 22.04 --name worker -m 4G -c 4 -d 8G
```

This step can take a few minutes as Multipass creates the new virtual machines. It's normal and expected.

Once the virtual machine has been created, you can run commands on it by opening a shell. For example:

```
multipass shell control-plane
```

This will behave as a local terminal session on the virtual machine, so you can run commands.

Install Canonical Kubernetes on both VMs with the following command:

```
sudo snap install --classic --edge k8s
```

2. Bootstrap your control plane node

Bootstrap the control plane node:

```
sudo k8s bootstrap
```

Canonical Kubernetes allows you to create two types of nodes: control plane and worker nodes. In this example, we're creating a worker node.

Generate the token required for the worker node to join the cluster by executing the following command on the control-plane node:

```
sudo k8s get-join-token worker --worker
```

A base64 token will be printed to your terminal. Keep it handy as you will need it for the next step.

Note: It's advisable to name the new node after the hostname of the worker node (in this case, the VM's hostname is worker).

---

3. Join the cluster on the worker node

To join the worker node to the cluster, run:

```
sudo k8s join-cluster <join-token>
```

After a few seconds, you should see: `Joined the cluster.`

4. View the status of your cluster

To see what we've accomplished in this tutorial:

If you created a control plane node, check that it joined successfully:

```
sudo k8s status
```

If you created a worker node, verify with this command:

```
sudo k8s kubectl get nodes
```

You should see that you've successfully added a worker or control plane node to your cluster.

Congratulations!

4. Remove Nodes and delete the VMs (Optional)

It is important to clean-up your nodes before tearing down the VMs.

Note: Purging a VM does not remove the node from your cluster.

Keep in mind the consequences of removing nodes:

Warning: Do not remove the leader node. If you have less than 3 nodes and you remove any node you will lose availability of your cluster.

To tear down the entire cluster, execute:

```
sudo k8s remove-node worker
sudo k8s remove-node control-plane
```

To delete the VMs from your system, two commands are needed:

```
multipass remove control-plane
multipass remove worker
multipass purge
```

2.3.3 Next Steps

- Discover how to enable and configure Ingress resources *Ingress*
- Keep mastering Canonical Kubernetes with kubectl *How to use kubectl*
- Explore Kubernetes commands with our *Command Reference Guide*
- Configure storage options *Storage*
- Master Kubernetes networking concepts *Networking*

2.4 Other documentation types

If you have a specific goal our *How-to guides* have more in-depth detail than tutorials and can be applied to a broader set of applications. They'll help you achieve an end-result but may require you to understand and adapt the steps to fit your specific requirements.

For a better understanding of how Canonical Kubernetes works and related topics such as security, our *Explanation section* helps you to expand your knowledge and get the most out of Kubernetes.

Finally, our *Reference section* is for when you need to check specific details or information such as the command reference or release notes.

CHAPTER

THREE

HOW-TO GUIDES

If you have a specific goal, but are already familiar with Kubernetes, our How-to guides are more specific and contain less background information. They'll help you achieve an end result but may require you to understand and adapt the steps to fit your specific requirements.

3.1 Installing Canonical Kubernetes

There's more than one way to install Canonical Kubernetes. You'll find links to the current How-to guides below.

3.1.1 Install Canonical Kubernetes from a snap

Canonical Kubernetes is packaged as a snap, available from the snap store for all supported platforms.

What you'll need

This guide assumes the following:

- You are installing on Ubuntu 22.04 or later, **or** another OS which supports snap packages (see snapd support)
- You have root or sudo access to the machine
- You have an internet connection
- The target machine has sufficient memory and disk space. To accommodate workloads, we recommend a system with ***at least*** 20G of disk space and 4G of memory.

Note: If you cannot meet these requirements, please see the *Installing* page for alternative options.

Check available channels (optional)

It is a good idea to check the available channels before installing the snap. Run the command:

```
snap info k8s
```

…which will output a list of currently available channels. See the *channels page* for an explanation of the different types of channel.

Install the snap

The snap can be installed with the snap command:

```
sudo snap install k8s --classic --channel=latest/edge
```

Note: The `latest/edge` channel is always under active development. This is where you will find the latest features but you may also experience instability.

Bootstrap the cluster

Installing the snap sets up all the parts required to run Kubernetes. The next step is to `bootstrap` the cluster to activate the services:

```
sudo k8s bootstrap
```

This command will output a message confirming local cluster services have been started.

Note: Additional configuration is possible by passing a YAML file. The various options are described in the *bootstrap reference documentation*.

Confirm the cluster is ready

It is recommended to ensure that the cluster initialises properly and is running with no issues. Run the command:

```
sudo k8s status --wait-ready
```

This command will wait until the cluster indicates it is ready and then display the current status. The command will time-out if the cluster does not reach a ready state.

3.1.2 Install with Multipass (Ubuntu/Mac/Windows)

Multipass https://multipass.run/ is a simple way to run Ubuntu in a virtual machine, no matter what your underlying OS. It is the recommended way to run Canonical Kubernetes on Windows and macOS systems, and is equally useful for running multiple instances of the `k8s` snap on Ubuntu too.

Install Multipass

Choose your OS for the install procedure

Ubuntu/Linux

Windows

macOS

Multipass is shipped as a snap for Ubuntu and other OSes which support the snap package system.

```
sudo snap install multipass
```

Windows users should download and install the Multipass installer from the website.

The latest version is available here https://multipass.run/download/windows, though you may wish to visit the Multipass website for more details.

Users running macOS should download and install the Multipass installer from the website.

The latest version is available here https://multipass.run/download/macos, though you may wish to visit the Multipass website for more details, including an alternate install method using `brew`.

Create an instance

The `k8s` snap will require a certain amount of resources, so the default settings for a Multipass VM aren't going to be suitable. Exactly what resources will be required depends on your use case. We recommend at least 4G of memory and 20G of disk space for each instance.

Open a terminal (or Shell on Windows) and enter the following command:

```
multipass launch 22.04 --name k8s-node --memory 4G --disk 20G --cpus 2
```

This command specifies:

- **22.04**: The Ubuntu image used as the basis for the instance
- **–name**: The name by which you will refer to the instance
- **–memory**: The memory to allocate
- **–disk**: The disk space to allocate
- **–cpus**: The number of CPU cores to reserve for this instance

For more details of creating instances with Multipass, please see the Multipass documentation about instance creation.

Access the created instance

To access the image you just created, run:

```
multipass shell k8s-node
```

This will immediately open a shell on the instance, so further commands you enter will be executed on the Ubuntu instance you created.

You can now use this terminal to install the `k8s` snap, following the standard *install instructions*, or following along with the *Getting started* tutorial if you are new to Canonical Kubernetes.

To end the shell session on the instance, enter:

```
exit
```

. . . and you will be returned to the original terminal session.

Stop/Remove the instance

The instance you created will keep running in the background until it is either stopped or the host computer is shut down. You can stop the running instance at any time by running:

```
multipass stop k8s-node
```

And it can be permanently removed with:

```
multipass delete k8s-node
multipass purge
```

3.1.3 Install Canonical Kubernetes in LXD

Canonical Kubernetes can also be installed inside an LXD container. This is a great way, for example, to test out clustered Canonical Kubernetes without the need for multiple physical hosts.

Installing LXD

You can install LXD via snaps:

```
sudo snap install lxd
sudo lxd init
```

Add the Canonical Kubernetes LXD profile

Canonical Kubernetes requires some specific settings to work within LXD (these are explained in more detail below). These can be applied using a custom profile. The first step is to create a new profile:

```
lxc profile create k8s
```

Once created, we'll need to add the rules. Get our pre-defined profile rules from GitHub and save them as `k8s.profile`.

```
wget https://raw.githubusercontent.com/canonical/k8s-snap/main/tests/integration/lxd-
↪profile.yaml -O k8s.profile
```

Note: For an explanation of the settings in this file, *see below*

To pipe the content of the file into the k8s LXD profile, run:

```
cat k8s.profile | lxc profile edit k8s
```

Remove the copied content from your directory:

```
rm k8s.profile
```

Start an LXD container for Canonical Kubernetes

We can now create the container that Canonical Kubernetes will run in.

```
lxc launch -p default -p k8s ubuntu:22.04 k8s
```

This command uses the `default` profile created by LXD for any existing system settings (networking, storage, etc.), before also applying the `k8s` profile - the order is important.

Install Canonical Kubernetes in an LXD container

First, we'll need to install Canonical Kubernetes within the container.

```
lxc exec k8s -- sudo snap install k8s --classic --channel=latest/edge
```

Note: Substitute your desired channel in the above command. Find the available channels with `snap info k8s` and see the *channels* explanation page for more details on channels, tracks and versions.

Access Canonical Kubernetes services within LXD

Assuming you accepted the default bridged networking when you initially setup LXD, there is minimal effort required to access Canonical Kubernetes services inside the LXD container.

Simply note the `eth0` interface IP address from the command:

```
lxc list k8s
```

```
+------+---------+----------------------+-----------------------------------------------+-
↪----------+-----------+
| NAME |  STATE  |         IPV4         |                     IPV6                      |␣
↪  TYPE    | SNAPSHOTS |
+------+---------+----------------------+-----------------------------------------------+-
↪----------+-----------+
| k8s  | RUNNING | 10.122.174.30 (eth0) | fd42:80c6:c3e:445a:216:3eff:fe8d:add9 (eth0) |␣
↪CONTAINER | 0         |
+------+---------+----------------------+-----------------------------------------------+-
↪----------+-----------+
```

and use this to access services running inside the container.

Expose services to the container

You'll need to expose the deployment or service to the container itself before you can access it via the LXD container's IP address. This can be done using `k8s kubectl expose`. This example will expose the deployment's port 80 to a port assigned by Kubernetes.

In this example, we will use Microbot as it provides a simple HTTP endpoint to expose. These steps can be applied to any other deployment.

First, initialize the k8s cluster with

```
lxc exec k8s -- sudo k8s bootstrap
```

Now, let's deploy Microbot (please note this image only works on `x86_64`).

```
lxc exec k8s -- sudo k8s kubectl create deployment microbot --image=dontrebootme/
↪microbot:v1
```

Then check that the deployment has come up.

```
lxc exec k8s -- sudo k8s kubectl get all
```

...should return output similar to:

```
NAME                              READY   STATUS    RESTARTS   AGE
pod/microbot-6d97548556-hchb7     1/1     Running   0          21m

NAME                         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
service/kubernetes           ClusterIP   10.152.183.1     <none>        443/TCP        21m

NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/microbot     1/1     1            1           21m

NAME                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/microbot-6d97548556     1         1         1       21m
```

Now that Microbot is up and running, let's make it accessible to the LXD container by using the `expose` command.

```
lxc exec k8s -- sudo k8s kubectl expose deployment microbot --type=NodePort --port=80 --
↪name=microbot-service
```

We can now get the assigned port. In this example, it's `32750`:

```
lxc exec k8s -- sudo k8s kubectl get service microbot-service
```

...returns output similar to:

```
NAME               TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
microbot-service   NodePort   10.152.183.188   <none>        80:32750/TCP   27m
```

With this, we can access Microbot from our host but using the container's address that we noted earlier.

```
curl 10.122.174.30:32750
```

Stop/Remove the container

The `k8s` container you created will keep running in the background until it is either stopped or the host computer is shut down. You can stop the running container at any time by running:

```
lxc stop k8s
```

And it can be permanently removed with:

```
lxc delete k8s
```

Explanation of custom LXD rules

boot.autostart: "true": Always start the container when LXD starts. This is needed to start the container when the host boots.

linux.kernel_modules: Comma separated list of kernel modules to load before starting the container

lxc.apparmor.profile=unconfined: Disable AppArmor. Allow the container to talk to a bunch of subsystems of the host (e.g. `/sys`). By default AppArmor will block nested hosting of containers, however Kubernetes needs to host Containers. Containers need to be confined based on their profiles thus we rely on confining them and not the hosts. If you can account for the needs of the Containers you could tighten the AppArmor profile instead of disabling it completely, as suggested in S.Graber's notes[1].

lxc.cap.drop=: Do not drop any capabilities[2]. For justification see above.

lxc.mount.auto=proc:rw sys:rw: Mount proc and sys rw. For privileged containers, lxc over-mounts part of /proc as read-only to avoid damage to the host. Kubernetes will complain with messages like `Failed to start ContainerManager open /proc/sys/kernel/panic: permission denied`

lxc.cgroup.devices.allow=a: "a" stands for "all." This means that the container is allowed to access all devices. It's a wildcard character indicating permission for all devices. For justification see above.

security.nesting: "true": Support running LXD (nested) inside the container.

security.privileged: "true": Runs the container in privileged mode, not using kernel namespaces[3],[4]. This is needed because hosted Containers may need to access for example storage devices (See comment in[5]).

3.2 Networking

Networking is a core part of a working Kubernetes cluster. These topics cover how to configure and use key capabilities of Canonical Kubernetes.

3.2.1 How to use default DNS

Canonical Kubernetes includes a default DNS (Domain Name System) which is essential for internal cluster communication. When enabled, the DNS facilitates service discovery by assigning each service a DNS name. When disabled, you can integrate a custom DNS solution into your cluster.

What you'll need

This guide assumes the following:

- You have root or sudo access to the machine.
- You have a bootstrapped Canonical Kubernetes cluster (see the *Getting Started* guide).

[1] https://stgraber.org/2012/05/04/
[2] https://stgraber.org/2014/01/01/lxc-1-0-security-features/
[3] https://unix.stackexchange.com/questions/177030/what-is-an-unprivileged-lxc-container/177031#177031
[4] http://blog.benoitblanchon.fr/lxc-unprivileged-container/
[5] https://wiki.ubuntu.com/LxcSecurity

Check DNS status

Find out whether DNS is enabled or disabled with the following command:

```
sudo k8s status
```

The default state for the cluster is `dns enabled`.

Enable DNS

To enable DNS, run:

```
sudo k8s enable dns
```

For more information on this command, run:

```
sudo k8s help enable
```

Configure DNS

Discover your configuration options by running:

```
sudo k8s get dns
```

You should see three options:

- `upstream-nameservers`: DNS servers used to forward known entries
- `cluster-domain`: the cluster domain name
- `service-ip`: the cluster IP to be assigned to the DNS service

Set a new DNS server IP for forwarding known entries:

```
sudo k8s set dns.upstream-nameservers=<new-ips>
```

Change the cluster domain name:

```
sudo k8s set dns.cluster-domain=<new-domain-name>
```

Assign a new cluster IP to the DNS service:

```
sudo k8s set dns.service-ip=<new-cluster-ip>
```

Replace <new-ip>, <new-domain-name>, and <new-cluster-ip> with the desired values for your DNS configuration.

Disable DNS

Canonical Kubernetes also allows you to disable the built-in DNS, if you desire a custom solution:

> **Warning:** Disabling DNS will disrupt internal cluster communication. Ensure a suitable custom DNS solution is in place before disabling. You can re-enable DNS at any point, and your cluster will return to normal functionality.```

```
sudo k8s disable dns
```

For more information on this command, execute:

```
sudo k8s help disable
```

3.2.2 How to use the default Network

Canonical Kubernetes includes a high-performance, advanced network plugin called Cilium. The network component allows cluster administrators to leverage software-defined networking to automatically scale and secure network policies across their cluster.

What you'll need

This guide assumes the following:

- You have root or sudo access to the machine.
- You have a bootstrapped Canonical Kubernetes cluster (see the *Getting Started* guide).

Check Network status

Find out whether Network is enabled or disabled with the following command:

```
sudo k8s status
```

The default state for the cluster is `network disabled`.

Enable Network

To enable Network, run:

```
sudo k8s enable network
```

For more information on the command, execute:

```
sudo k8s enable network --help
```

Configure Network

It is not possible to reconfigure the network on a running cluster as this will lead to unreachable pods/services and nodes. Any configuration options the CNI needs to be aware of (e.g. pod and service CIDR, IPv6 support) are set during the cluster bootstrap ("k8s bootstrap" command).

Check Network details

Let's look at the detailed status of the network as reported by Cilium.

First, find the name of the Cilium pod:

```
sudo k8s kubectl get pod -n kube-system -l k8s-app=cilium
```

Once you have the name of the pod, run the following command to see Cilium's status:

```
sudo k8s kubectl exec -it cilium-97vcw -n kube-system -c cilium-agent -- cilium status
```

You should see a wide range of metrics and configuration values for your cluster.

Disable Network

You can `disable` the built-in network:

> **Warning:** If you have an active cluster, disabling Network may impact external access to services within your cluster. Ensure that you have alternative configurations in place before disabling Network.

```
sudo k8s disable network
```

For more information on this command, run:

```
sudo k8s disable network --help
```

3.2.3 How to use default Ingress

Canonical Kubernetes allows you to configure Ingress into your cluster. When enabled, it tells your cluster how external HTTP and HTTPS traffic should be routed to its services.

What you'll need

This guide assumes the following:

- You have root or sudo access to the machine
- You have a bootstrapped Canonical Kubernetes cluster (see the *Getting Started* guide).

Check Ingress status

Find out whether Ingress is enabled or disabled with the following command:

```
sudo k8s status
```

The default state for the cluster is `ingress disabled`.

Enable Ingress

To enable Ingress, run:

```
sudo k8s enable ingress
```

For more information on the command, execute:

```
sudo k8s help enable
```

Configure Ingress

Discover your configuration options by running:

```
sudo k8s get ingress
```

You should see three options:

- `default-tls-secret`: Name of the TLS (Transport Layer Security) Secret in the kube-system namespace that will be used as the default Ingress certificate
- `enable-proxy-protocol`: If set, proxy protocol will be enabled for the Ingress

TLS Secret

You can create a TLS secret by following the official Kubernetes documentation. Note: remember to use `sudo k8s kubectl` (See the *kubectl-guide*).

Tell Ingress to use your new Ingress certificate:

```
sudo k8s set ingress.default-tls-secret=<new-default-tls-secret>
```

Replace `<new-default-tls-secret>` with the desired value for your Ingress configuration.

Proxy Protocol

Enabling the proxy protocol allows passing client connection information to the backend service.

Consult the official Kubernetes documentation on the proxy protocol.

Use the following command to enable the proxy protocol:

```
sudo k8s set ingress.enable-proxy-protocol=<new-enable-proxy-protocol>
```

Adjust the value of `<new-enable-proxy-protocol>` with your proxy protocol requirements.

Disable Ingress

You can `disable` the built-in ingress:

> **Warning:** Disabling Ingress may impact external access to services within your cluster. Ensure that you have alternative configurations in place before disabling Ingress.

```
sudo k8s disable ingress
```

For more information on this command, run:

```
sudo k8s help disable
```

3.2.4 How to use the default load-balancer

Canonical Kubernetes includes a default load-balancer. As this is not an essential service for all deployments, it is not enabled by default. This guide explains how to configure and enable the load-balancer.

What you'll need

This guide assumes the following:

- You have root or sudo access to the machine.
- You have a bootstraped Canonical Kubernetes cluster (see the *Getting Started* guide).

Check the status and configuration

Find out whether DNS is enabled or disabled with the following command:

```
sudo k8s status
```

The load-balancer is not enabled by default, it won't be listed on the status output unless it has been subsequently enabled.

To check the current configuration of the load-balancer, run the following:

```
sudo k8s get load-balancer
```

This should output a list of values like this:

- `cidrs` - a list containing cidr or IP address range definitions of the pool of IP addresses to use
- `l2-mode` - whether L2 mode (failover) is turned on
- `l2-interfaces` - optional list of interfaces to announce services over (defaults to all)
- `bgp-mode` - whether BGP mode is active.
- `bgp-local-asn` - the local Autonomous System Number (ASN)
- `bgp-peer-address` - the peer address
- `bgp-peer-asn` - ASN of the peer network
- `bgp-peer-port` - port used on the BGP peer

These values are configured using the `k8s set`command, e.g.:

```
sudo k8s set load-balancer.l2-mode=true
```

Note that for the BGP mode, it is necessary to set ***all*** the values simultaneously. E.g.

```
sudo k8s set load-balancer.bgp-mode=true load-balancer.bgp-local-asn=64512 load-balancer.
↪bgp-peer-address=10.0.10.55/32 load-balancer.bgp-peer-asn=64512 load-balancer.bgp-peer-
↪port=7012
```

Enable the load-balancer

To enable the load-balancer, run:

```
sudo k8s enable load-balancer
```

You can now confirm it is working by running:

```
sudo k8s status
```

Disable the load-balancer

The default load-balancer can be disabled again with:

```
sudo k8s disable load-balancer
```

3.3 How to use default storage

Canonical Kubernetes offers a local-storage option to quickly set up and run a cluster, especially for single-node support. This guide walks you through enabling and configuring this feature.

3.3.1 What you'll need

This guide assumes the following:

- You have root or sudo access to the machine
- You have a bootstrapped Canonical Kubernetes cluster (see the *getting-started-guide*)

3.3.2 Enable Local Storage

When bootstrapping the snap, the storage feature is not enabled by default. To enable it, execute the following command:

```
sudo k8s enable local-storage
```

3.3.3 Configure Local Storage

While the storage option comes with sensible defaults, you can customise it to meet your requirements. Obtain the current configuration by running:

```
sudo k8s get local-storage
```

You can modify the configuration using the `set` command. For example, to change the local storage path:

```
sudo k8s set local-storage.local-path=/path/to/new/folder
```

The local-storage feature provides the following configuration options:

- `local-path`: path where the local files will be created.
- `reclaim-policy`: set the reclaim policy of the persistent volumes provisioned. It should be one of "Retain", "Recycle", or "Delete".
- `default`: set the local-storage storage class to be the default. If this flag is not set and the cluster already has a default storage class it is not changed. If this flag is not set and the cluster does not have a default class set then the class from the local-storage becomes the default.

3.3.4 Disable Local Storage

The local storage option is only suitable for single-node clusters and development environments as it has no multi node data replication. For a production environment you may want a more sophisticated storage solution. To disable local-storage, run:

```
sudo k8s disable local-storage
```

Disabling storage only removes the CSI driver. The persistent volume claims will still be available and your data will remain on disk.

3.4 How to use an external datastore

Canonical Kubernetes supports using an external datastore such as etcd instead of the bundled dqlite datastore. This guide walks you through configuring an external etcd datastore.

3.4.1 What you'll need

This guide assumes the following:

- You have root or sudo access to the machine
- You have an external etcd cluster
- You have installed the Canonical Kubernetes snap (see How-to *Install Canonical Kubernetes from a snap*).
- You have not bootstrapped the Canonical Kubernetes cluster yet

Warning: The selection of the backing datastore can only be changed during the bootstrap process. There is no migration path between the bundled dqlite and the external datastores.

3.4.2 Adjust the bootstrap configuration

To use an external datastore, a configuration file that contains the required datastore parameters needs to be provided to the bootstrap command. Create a configuration file and insert the contents below while replacing the placeholder values based on the configuration of your etcd cluster.

```
datastore: external
datastore-url: "<etcd-member-addresses>"
datastore-ca-crt: |
  <etcd-root-ca-certificate>
datastore-client-crt: |
  <etcd-client-certificate>
datastore-client-key: |
  <etcd-client-key>
```

- `datastore-url` expects a comma seperated list of addresses (e.g. `https://10.42.254.192:2379,https://10.42.254.193:2379,https://10.42.254.194:2379`)
- `datastore-ca-crt` expects a certificate for the CA in PEM format
- `datastore-client-crt` expects a certificate that's signed by the root CA for the client in PEM format
- `datastore-client-key` expects a key for the client in PEM format

Note: `datastore-ca-crt`, `datastore-client-crt` and `datastore-client-key` options can be omitted if the etcd cluster is not configured to use secure connections.

3.4.3 Bootstrap the cluster

The next step is to bootstrap the cluster with our configuration file:

```
sudo k8s bootstrap --file /path/to/config.yaml
```

Note: The datastore can only be configured through the `--file` file option, and is not available in interactive mode.

3.4.4 Confirm the cluster is ready

It is recommended to ensure that the cluster initialises properly and is running without issues. Run the command:

```
sudo k8s status --wait-ready
```

This command will wait until the cluster is ready and then display the current status. The command will time-out if the cluster does not reach a ready state.

3.5 Configure proxy settings for K8s

Canonical Kubernetes packages a number of utilities (eg curl, helm) which need to fetch resources they expect to find on the internet. In a constrained network environment, such access is usually controlled through proxies.

On Ubuntu and other Linux operating systems, proxies are configured through system-wide environment variables defined in the `/etc/environment` file.

3.5.1 Adding proxy configuration for the k8s snap

Edit the `/etc/environment` file and add the relevant URLs

Note: It is important to add whatever address ranges are used by the cluster itself to the `NO_PROXY` and `no_proxy` variables.

For example, assume we have a proxy running at `http://squid.internal:3128` and we are using the networks `10.0.0.0/8`,`192.168.0.0/16` and `172.16.0.0/12`. We would edit the environment (`/etc/environment`) file to include these lines:

```
HTTPS_PROXY=http://squid.internal:3128
HTTP_PROXY=http://squid.internal:3128
NO_PROXY=10.0.0.0/8,192.168.0.0/16,127.0.0.1,172.16.0.0/12
https_proxy=http://squid.internal:3128
http_proxy=http://squid.internal:3128
no_proxy=10.0.0.0/8,192.168.0.0/16,127.0.0.1,172.16.0.0/12
```

Note that you may need to restart for these settings to take effect.

Note: The **10.152.183.0/24** CIDR needs to be covered in the juju-no-proxy list as it is the Kubernetes service CIDR. Without this any pods will not be able to reach the cluster's kubernetes-api. You should also exclude the range used by pods (which defaults to **10.1.0.0/16**) and any required local networks.

3.5.2 Adding proxy configuration for the k8s charms

Proxy confiuration is handled by Juju when deploying the `k8s` charms. Please see the *documentation for adding proxy configuration via Juju*.

3.6 How to contribute to Canonical Kubernetes

Canonical Kubernetes is proudly open source, published under the GPLv3 license. We welcome and encourage contributions to the code and the documentation. See the *community page* for ways to get in touch and provide feedback.

3.6.1 Contribute to the code

Canonical Kubernetes is shipped as a snap package. To contribute to the code, you should first make sure you can build and test the snap locally.

Build the snap

To build the snap locally, you will need the following:

- The latest, snap-based version of LXD (see the install guide here)
- The Snapcraft build tool, for building the snap (see the Snapcraft documentation).

Clone the GitHub repository for the k8s snap and then open a terminal in that directory. Run the command:

```
snapcraft --use-lxd
```

This will launch an LXD container and use it to build a version of the snap. This will take some time as the build process fetches dependencies, stages the 'parts' of the snap and creates the snap package itself. The snap itself will be fetched from the build environment and placed in the local project directory. Note that the LXD container used for building will be stopped, but not deleted. This is in case there were any errors or artefacts you may wish to inspect.

Install the snap

The snap can then be installed locally by using the '–dangerous' option. This is a safeguard to make sure the user is aware that the snap is not signed by the snap store, and is not confined:

```
sudo snap install k8s_v1.29.2_multi.snap --dangerous --classic
```

Note: You will not be able to install this snap if there is already a k8s snap installed on your system.

Once you have verified the current snap build works, it can be removed with:

```
sudo snap remove k8s --purge
```

The `purge` option is recommended when iterating over code changes, as it also removes all the installed artefacts which may be associated with the snap.

Now you can iterate over changes to the snap, rebuild and test.

As noted previously, the LXD container used for building is not removed and will be reused by subsequent build instructions. When you are satisfied it is no longer needed, this container can be removed:

```
lxc delete snapcraft-k8s
```

Contribute changes

We welcome any improvements and bug-fixes to the Canonical Kubernetes code. Once you have tested your changes, please make a pull request on the code repository and we will review it as soon as possible.

3.6.2 Contribute to the documentation

Our aim is to provide easy-to-understand documentation on all aspects of Canonical Kubernetes, so we greatly appreciate your feedback and contributions. See our *community page* for ways of getting in touch.

The source of the documentation and the system used to build it are included in the main repository for the Canonical Kubernetes snap.

Documentation framework

This documentation has adopted the Diátaxis framework. You can read more about it on the Diátaxis website. In essence though, this guides the way we categorise and write our documentation. You can see there are four main categories of documentation:

- **Tutorials** for guided walkthroughs
- **How to** pages for specific tasks and goals
- **Explanation** pages which give background reasons and, well, explanations
- **Reference**, where you will find the commands, the roadmap, etc.

Every page of documentation should fit into one of those categories. If it doesn't you may consider if it is actually two pages (e.g. a How to *and* an explanation).

We have included some tips and outlines of the different types of docs we create to help you get started:

- Tutorial template
- How to template
- Explanation template
- Reference template

Small changes

If you are simply correcting a typo or updating a link, you can follow the 'Edit this page on GitHub' link on any page and it will take you to the online editor to make your change. You will still need to raise a pull request and explain your change to get it reviewed.

Myst, Markdown and Sphinx

We use the Sphinx documentation tools to actually build the documentation. You will find all the Sphinx tooling in the `docs/tools` directory.

Although Sphinx is normally associated with the `ReSTructured text` format, we write all our documentation in Markdown to make it easier for humans to work with. There are a few extra things that come with this - certain features need to be specially marked up (e.g. admonitions) to be processed properly. There is a guide to using `Myst` (which is a Markdown extension for Sphinx) directives and formatting contained in the _parts directory of the docs.

Local testing

To test your changes locally, you can build a local version of the documentation. Open a terminal and go to the `/docs/tools` directory. From there you can run the command:

```
make run
```

This will create a local environment, install all the dependencies and build the docs. The output will then be served locally - check the output for the URL. Using the `run` option means that the docs will automatically be regenerated when you change any of the source files too (though remember to press `F5` in your browser to reload the page without caching)!

3.7 How to get support

Support for Canonical Kubernetes is available in a variety of ways:

- Engagement with the *Canonical Kubernetes Community*
- Understanding common *Troubleshooting Techniques*
- Professional support services with Ubuntu Support

3.8 Other documentation types

Our Reference section is for when you need to check specific details or information such as the command reference or release notes.

Alternatively, the *Tutorials section* contains step-by-step tutorials to help guide you through exploring and using Canonical Kubernetes.

For a better understanding of how Canonical Kubernetes works and related topics such as security, our *Explanation section* helps you expand your knowledge and get the most out of Kubernetes.

Finally, our *Reference section* is for when you need to check specific details or information such as the command reference or release notes.

CHAPTER

FOUR

EXPLANATION

For a better understanding of how Canonical Kubernetes works and related topics such as security, these pages will help expand your knowledge and get the most out of Kubernetes.

4.1 What is Canonical Kubernetes?

At its core, Canonical Kubernetes is a full implementation of upstream Kubernetes delivered in a compact, secure, reliable snap package. As the upstream Kubernetes services are not all that is required for a fully functional cluster, additional services and features are built in. You can deploy the snap and have a single-node cluster up and running in minutes.

4.1.1 Why a snap?

Snaps are self-contained, simple to install, secure, cross-platform, and dependency-free. They can be installed on any Linux system which supports the `snapd` service (see the snapd documentation for more information). Security and robustness are their key features, alongside being easy to install, easy to maintain and easy to upgrade.

4.1.2 What else comes with it?

In addition to the upstream Kubernetes services, Canonical Kubernetes also includes:

- a DNS service for the node
- a CNI for the node/cluster
- a simple storage provider
- an ingress provider
- a load-balancer
- a gateway API controller
- a metrics server

4.1.3 Where can I install it?

The Canonical Kubernetes snap can be installed on a Linux OS, wherever it may be: run it in several local containers or VMs for example, or use it on public/private cloud instances. For deploying with Juju, a machine charm to deploy the snap is also available.

4.1.4 Can I use it to make a cluster?

Yes. Canonical Kubernetes is designed to be eminently scalable. You can start with a single node and add more as and when the need arises. Scale up or down at any time. Systems with more than three nodes will automatically become Highly Available.

4.1.5 Does it come with support?

Each and every user will be supported by the community. For a more detailed look at what that entails, please see our *Community page*. If you need a greater level of support, Canonical provides Ubuntu Pro, a comprehensive subscription for your open-source software stack. For more support options, visit the Ubuntu support page.

4.1.6 Next steps

- Try it now! Jump over to the *Getting started tutorial*

4.2 Channels

Canonical Kubernetes uses the concept of `channels` to make sure you always get the version of Kubernetes you are expecting, and that future upgrades can be handled with minimum, if any, disruption.

4.2.1 Choosing the right channel

When installing or updating Canonical Kubernetes you can (and should in most cases) specify a channel. The channel specified is made up of two components; the **track** and the **risk level**.

The track matches the minor version of upstream Kubernetes. For example, specifying the `1.30` track will match upstream releases of the same minor version ("1.30.0", "1.30.1", "1.30.x" etc.). Releases of Canonical Kubernetes closely follow the upstream releases and usually follow within 24 hours.

The 'risk level' component of the channel is one of the following:

- **`stable`**: Matches upstream stable releases
- **`candidate`**: Holds the release candidates of the snap
- **`beta`**: Tracks the beta releases - expect bugs
- **`edge`**: Experimental release including upstream alpha releases

Note that for each track, not all risk levels are guaranteed to be available. For example, there may be a new upstream version in development which only has an `edge` level. For a mature release, there may no longer be any `beta` or `candidate`. In these cases, if you specify a risk level which has no releases for that track the snap system will choose the closest available release with a lower risk level. Whatever risk level specified is the **maximum** risk level of the snap that will be installed - if you choose `candidate` you will never get `edge` for example.

For all snaps, you can find out what channels are available by running the `info` command, For example:

```
snap info k8s
```

More information can be found in the Snapcraft documentation

4.2.2 Updates and switching channels

Updates for upstream patch releases will happen automatically by default. For example, if you have selected the channel `1.30/stable`, your snap will refresh itself regularly keeping your cluster up-to-date with the latest patches. For deployments where this behaviour is undesirable you are given the option to postpone, schedule or even block automatic updates. The Snap refreshes documentation page outlines how to configure these options.

To change the channel of an already installed snap, the `refresh` command can be used:

```
sudo snap refresh k8s --channel=<new-channel>
```

> **Warning:** Changing the channel of an installed snap could result in loss of service. Please check any release notes or upgrade guides first!

4.2.3 Which channel is right for me?

Choosing the most appropriate channel for your needs depends on a number of factors. We can give some general guidance for the following cases:

- **I want to always be on the latest stable version matching a specific upstream K8s release (recommended).**

Specify the release, for example: `--channel=1.30/stable`.

- **I want to test-drive a pre-stable release**

Use `--channel=<next_release>/edge` for alpha releases.

Use `--channel=<next_release>/beta` for beta releases.

Use `--channel=<next_release>/candidate` for candidate releases.

- **I am waiting to test a bug fix on Canonical Kubernetes**

Use `--channel=<release>/edge`.

- **I am waiting for a bug fix from upstream Kubernetes**

Use `--channel=<release>/candidate`.

4.3 Clustering

Kubernetes clustering allows you to manage a group of hosts as a single entity. This enables applications to be deployed across a cluster of machines without tying them specifically to one host, providing high availability and scalability. In Canonical Kubernetes the addition of `k8sd` to the Kubernetes ecosystem introduces enhanced capabilities for cluster coordination and management.

4.3.1 Kubernetes Cluster Topology

A Kubernetes cluster consists of at least one control plane node and multiple worker nodes. Each node is a server (physical or virtual) that runs Kubernetes components. In Canonical Kubernetes, the components are bundled inside the `k8s-snap`. The cluster's topology divides responsibilities between the control plane node(s) and the worker nodes, ensuring efficient management and scheduling of workloads.

This is the overview of a Canonical Kubernetes cluster:

4.3.2 The Role of `k8sd` in Kubernetes Clustering

`k8sd` plays a vital role in the Canonical Kubernetes architecture, enhancing the functionality of both the Control Plane and Worker nodes through the use of microcluster. This component simplifies cluster management tasks, such as adding or removing nodes and integrating them into the cluster. It also manages essential features like DNS and networking within the cluster, streamlining the entire process for a more efficient operation.

4.3.3 Integration into the Kubernetes Cluster Topology

For Canonical Kubernetes, the detailed view of the two types of node is as follows:

Control Plane Node

The control plane node orchestrates the cluster, making decisions about scheduling, deployment and management. With the addition of `k8sd`, the control plane node's components include:

- **API Server (kube-apiserver)**: Acts as the front-end for the Kubernetes control plane. It exposes the Kubernetes API and is the central management entity through which all components and external users interact.
- **Scheduler (kube-scheduler)**: Responsible for allocating pods to nodes based on various criteria such as resource availability and constraints.
- **Controller Manager (kube-controller-manager)**: Runs controller processes that regulate the state of the cluster, ensuring the desired state matches the observed state.
- **k8s-dqlite**: A fast, embedded, persistent in-memory key-value store with Raft consensus used to store all cluster data.
- **k8sd**: Implements and exposes the operations functionality needed for managing the Kubernetes cluster.

Worker Node

Worker nodes are responsible for running the applications and workloads. Worker nodes, can interact with the `k8sd` API, gaining capabilities to manage its entire life-cycle. Their components include:

- **Local API Server Proxy**: This component forwards requests to the control plane nodes.
- **Kubelet**: Communicates with the control plane node and manages the containers running on the machine according to the configurations provided by the user.
- **Kube-Proxy (kube-proxy)**: Manages network communication within the cluster.
- **Container Runtime**: The software responsible for running containers. In Canonical Kubernetes the runtime is `containerd`.

4.4 Ingress

In Kubernetes, understanding how inbound traffic is managed inside of your cluster can be complex. This explanation provides you with the essentials to successfully manage your Canonical Kubernetes cluster.

4.4.1 Network

When you install Canonical Kubernetes, the default Network is automatically enabled. This is also a requirement for the default Ingress to function.

Since upstream Kubernetes comes without a network provider, it requires the use of a network plugin. This network plugin facilitates communication between pods, services, and external resources, ensuring smooth traffic flow within the cluster. The current implementation of Canonical Kubernetes leverages a widely adopted CNI (Container Network Interface) called Cilium. If you wish to use a different network plugin the implementation and configuration falls under your responsibility.

Learn how to use the Canonical Kubernetes default network in the *networking HowTo guide*.

4.4.2 Kubernetes Pods and Services

In Kubernetes, the smallest unit is a pod, which encapsulates application containers. Since pods are ephemeral and their IP addresses change when destroyed and restarted, they are exposed through services. Services offer a stable network interface by providing discoverable names and load balancing functionality for managing a set of pods. For further details on Kubernetes Services, refer to the upstream Kubernetes Service documentation.

4.4.3 Ingress

Ingress is a Kubernetes resource that manages external access by handling both HTTP and HTTPS traffic to services within your cluster. Traffic routed through the Ingress is directed to a service, which in turn forwards it to the relevant pod running the desired application within a container.

The Ingress resource lets you define rules on how traffic should get handled. Refer to the Kubernetes documentation on Ingress rules for up to date information on the available rules and their implementation.

While the Ingress resource manages the routing rules for the incoming traffic, the Ingress Controller is responsible for implementing those rules by configuring the underlying networking infrastructure of the cluster. Ingress does not work without an Ingress Controller.

The Ingress Controller also serves as a layer 7 (HTTP/HTTPS) load balancer that routes traffic from outside of your cluster to services inside of your cluster. Please do not confuse this with the Kubernetes Service LoadBalancer type which operates at layer 4 and routes traffic directly to individual pods.

With Canonical Kubernetes, enabling Ingress is easy: See the *default Ingress guide*. Once enabled, you will have a working Ingress Controller in your cluster.

The underlying mechanism provided by default is currently Cilium. However, it should always be operated through the provided CLI rather than directly. This way, we can provide the best experience for future cluster maintenance and upgrades.

If your cluster requires different Ingress Controllers, the responsibility of implementation falls upon you.

You will need to create the Ingress resource, outlining rules that direct traffic to your application's Kubernetes service.

4.5 Security

This page provides an overview of various aspects of security to be considered when operating a cluster with **Canonical Kubernetes**. To consider security properly, this means not just aspects of Kubernetes itself, but also how and where it is installed and operated.

A lot of important aspects of security therefore lie outside the direct scope of **Canonical Kubernetes**, but links for further reading are provided.

4.5.1 Security of the snap/executable

As detailed in the snap documentation, an application installed from a snap is inherently more secure than a traditionally installed application. Snap-based applications are installed into a sandboxed, self contained environment which restricts its ability to interact with the rest of user space.

4.5.2 Security of the OCI images

Canonical Kubernetes relies on OCI standard images published as `rocks` to deliver the services which run and facilitate the operation of the Kubernetes cluster. The use of Rockcraft and `rocks` gives Canonical a way to maintain and patch images to remove vulnerabilities at their source, which is fundamental to our commitment to a sustainable Long Term Support(LTS) release of Kubernetes and overcoming the issues of stale images with known vulnerabilities. For more information on how these images are maintained and published, see the Rockcraft documentation.

4.5.3 Kubernetes Security

The Kubernetes cluster deployed by Canonical Kubernetes can be secured using any of the methods and options described by the upstream Kubernetes Security Documentation.

Canonical Kubernetes enables RBAC (Rules Based Access Control) by default.

4.5.4 Cloud security

If you are deploying **Canonical Kubernetes** on public or private cloud instances, anyone with credentials to the cloud where it is deployed may also have access to your cluster. Describing the security mechanisms of these clouds is out of the scope of this documentation, but you may find the following links useful.

- Amazon Web Services https://aws.amazon.com/security/
- Google Cloud Platform https://cloud.google.com/security/
- Metal As A Service(MAAS) https://maas.io/docs/snap/3.0/ui/hardening-your-maas-installation
- Microsoft Azure https://docs.microsoft.com/en-us/azure/security/azure-security
- VMWare VSphere https://www.vmware.com/security/hardening-guides.html

4.5.5 Security Compliance

As with previously released Kubernetes software from Canonical, we aim to satisfy the needs of various security compliance standards. This is a process that will take some time however. Please watch out for future announcements and check the *roadmap* for current areas of work.

4.6 Other documentation types

If you are just getting started, the *Tutorials section* contains step-by-step tutorials to help guide you through exploring and using Canonical Kubernetes.

If you have a specific goal our *How-to guides* have more in-depth detail than tutorials and can be applied to a broader set of applications. They'll help you achieve an end-result but may require you to understand and adapt the steps to fit your specific requirements.

Finally, our *Reference section* is for when you need to check specific details or information such as the command reference or release notes.

CHAPTER

FIVE

REFERENCE

Our Reference section is for when you need to check specific details or information such as the command reference or release notes.

5.1 Release notes

5.1.1 Rolling preview release

In advance of a GA release of Canonical Kubernetes, you can still install and try out the newest distribution of Kubernetes.

You need two commands to get a single node cluster, one for installation and another for cluster bootstrap. You can try it out now on your console by installing the k8s snap from the beta channel:

```
sudo snap install k8s --channel=1.30-classic/beta --classic
sudo k8s bootstrap
```

Currently Canonical Kubernetes is working towards general availability, but you can install it now to try:

- **Clustering** - need high availability or just an army of worker nodes? Canonical Kubernetes is emminently scaleable, see the *tutorial on adding more nodes*.
- **Networking** - Our built-in network component allows cluster administrators to automatically scale and secure network policies across the cluster. Find out more in our *how-to guides*.
- **Observability** - Canonical Kubernetes ships with COS Lite, so you never need to wonder what your cluster is actually doing. See the *observability documentation* for more details.

Follow along with the *tutorial* to get started!

5.2 Command reference

These are the commands provided by the k8s snap:

5.2.1 k8s

Canonical Kubernetes CLI

Options

```
  -h, --help   help for k8s
```

5.2.2 k8s bootstrap

Bootstrap a new Kubernetes cluster

Synopsis

Generate certificates, configure service arguments and start the Kubernetes services.

```
k8s bootstrap [flags]
```

Options

```
      --address string         microcluster address, defaults to the node IP address
      --file string            path to the YAML file containing your custom cluster␣
↪bootstrap configuration. Use '-' to read from stdin.
  -h, --help                   help for bootstrap
      --interactive            interactively configure the most important cluster options
      --name string            node name, defaults to hostname
      --output-format string   set the output format to one of plain, json or yaml␣
↪(default "plain")
```

5.2.3 k8s config

Generate an admin kubeconfig that can be used to access the Kubernetes cluster

```
k8s config [flags]
```

Options

```
  -h, --help            help for config
      --server string   custom cluster server address
```

Options inherited from parent commands

```
      --output-format string   set the output format to one of plain, json or yaml␣
↪(default "plain")
      --timeout duration       the max time to wait for the command to execute (default␣
↪1m30s)
```

5.2.4 k8s disable

Disable core cluster features

Synopsis

Disable one of network, dns, gateway, ingress, local-storage, load-balancer.

```
k8s disable <feature> ... [flags]
```

Options

```
  -h, --help                   help for disable
      --output-format string   set the output format to one of plain, json or yaml␣
↪(default "plain")
```

5.2.5 k8s enable

Enable core cluster features

Synopsis

Enable one of network, dns, gateway, ingress, local-storage, load-balancer.

```
k8s enable <feature> ... [flags]
```

Options

```
  -h, --help                   help for enable
      --output-format string   set the output format to one of plain, json or yaml␣
↪(default "plain")
```

5.2.6 k8s get-join-token

Create a token for a node to join the cluster

```
k8s get-join-token <node-name> [flags]
```

Options

```
  -h, --help     help for get-join-token
      --worker   generate a join token for a worker node
```

5.2.7 k8s get

Get cluster configuration

Synopsis

Show configuration of one of network, dns, gateway, ingress, local-storage, load-balancer.

```
k8s get <feature.key> [flags]
```

Options

```
  -h, --help                   help for get
      --output-format string   set the output format to one of plain, json or yaml␣
↪(default "plain")
```

5.2.8 k8s join-cluster

Join a cluster using the provided token

```
k8s join-cluster <join-token> [flags]
```

Options

```
      --address string         microcluster address, defaults to the node IP address
      --file string            path to the YAML file containing your custom cluster join␣
↪configuration. Use '-' to read from stdin.
  -h, --help                   help for join-cluster
      --name string            node name, defaults to hostname
      --output-format string   set the output format to one of plain, json or yaml␣
↪(default "plain")
```

5.2.9 k8s kubectl

Integrated Kubernetes kubectl client

```
k8s kubectl [flags]
```

Options

```
  -h, --help   help for kubectl
```

5.2.10 k8s remove-node

Remove a node from the cluster

```
k8s remove-node <node-name> [flags]
```

Options

```
      --force                  forcibly remove the cluster member
  -h, --help                   help for remove-node
      --output-format string   set the output format to one of plain, json or yaml␣
↪(default "plain")
```

5.2.11 k8s set

Set cluster configuration

Synopsis

Configure one of network, dns, gateway, ingress, local-storage, load-balancer. Use `k8s get` to explore configuration options.

```
k8s set <feature.key=value> ... [flags]
```

Options

```
  -h, --help                   help for set
      --output-format string   set the output format to one of plain, json or yaml␣
↪(default "plain")
```

5.2.12 k8s status

Retrieve the current status of the cluster

```
k8s status [flags]
```

Options

```
  -h, --help                   help for status
      --output-format string   set the output format to one of plain, json or yaml␣
↪(default "plain")
      --wait-ready             wait until at least one cluster node is ready
```

5.3 Bootstrap configuration file reference

A YAML file can be supplied to the `k8s bootstrap` command to configure and customise the cluster. This reference section provides the format of this file by listing all available options and their details. See below for an example.

5.3.1 Format Specification

cluster-config.network

Type: `object` **Required:** `No`

Configuration options for the network feature

cluster-config.network.enabled

Type: `bool` **Required:** `No`

Determines if the feature should be enabled. If omitted defaults to `true`

cluster-config.dns

Type: `object` **Required:** `No`

Configuration options for the dns feature

cluster-config.dns.enabled

Type: `bool` **Required:** `No`

Determines if the feature should be enabled. If omitted defaults to `true`

cluster-config.dns.cluster-domain

Type: `string` **Required:** `No`

Sets the local domain of the cluster. If omitted defaults to `cluster.local`

cluster-config.dns.service-ip

Type: `string` **Required:** `No`

Sets the IP address of the dns service. If omitted defaults to the IP address of the Kubernetes service created by the feature.

Can be used to point to an external dns server when feature is disabled.

cluster-config.dns.upstream-nameservers

Type: `list[string]` **Required:** `No`

Sets the upstream nameservers used to forward queries for out-of-cluster endpoints. If omitted defaults to `/etc/resolv.conf` and uses the nameservers of the node.

cluster-config.ingress

Type: `object` **Required:** `No`

Configuration options for the ingress feature

cluster-config.ingress.enabled

Type: `bool` **Required:** `No`

Determines if the feature should be enabled. If omitted defaults to `false`

cluster-config.ingress.default-tls-secret

Type: `string` **Required:** `No`

Sets the name of the secret to be used for providing default encryption to ingresses.

Ingresses can specify another TLS secret in their resource definitions, in which case the default secret won't be used.

cluster-config.ingress.enable-proxy-protocol

Type: `bool` **Required:** `No`

Determines if the proxy protocol should be enabled for ingresses. If omitted defaults to `false`

cluster-config.load-balancer

Type: `object` **Required:** `No`

Configuration options for the load-balancer feature

cluster-config.load-balancer.enabled

Type: `bool` **Required:** `No`

Determines if the feature should be enabled. If omitted defaults to `false`

cluster-config.load-balancer.cidrs

Type: `list[string]` **Required:** `No`

Sets the CIDRs used for assigning IP addresses to Kubernetes services with type `LoadBalancer`.

cluster-config.load-balancer.l2-mode

Type: `bool` **Required:** `No`

Determines if L2 mode should be enabled. If omitted defaults to `false`

cluster-config.load-balancer.l2-interfaces

Type: `list[string]` **Required:** `No`

Sets the interfaces to be used for announcing IP addresses through ARP. If omitted all interfaces will be used.

cluster-config.load-balancer.bgp-mode

Type: `bool` **Required:** `No`

Determines if BGP mode should be enabled. If omitted defaults to `false`

cluster-config.load-balancer.bgp-local-asn

Type: `int` **Required:** `Yes if bgp-mode is true`

Sets the ASN to be used for the local virtual BGP router.

cluster-config.load-balancer.bgp-peer-address

Type: `string` **Required:** `Yes if bgp-mode is true`

Sets the IP address of the BGP peer.

cluster-config.load-balancer.bgp-peer-asn

Type: `int` **Required:** `Yes if bgp-mode is true`

Sets the ASN of the BGP peer.

cluster-config.load-balancer.bgp-peer-port

Type: `int` **Required:** `Yes if bgp-mode is true`

Sets the port of the BGP peer.

cluster-config.local-storage

Type: `object` **Required:** `No`

Configuration options for the local-storage feature

cluster-config.local-storage.enabled

Type: `bool` **Required:** `No`

Determines if the feature should be enabled. If omitted defaults to `false`

cluster-config.local-storage.local-path

Type: `string` **Required:** `No`

Sets the path to be used for storing volume data. If omitted defaults to `/var/snap/k8s/common/rawfile-storage`

cluster-config.local-storage.reclaim-policy

Type: `string` **Required:** `No` **Possible Values:** `Retain | Recycle | Delete`

Sets the reclaim policy of the storage class. If omitted defaults to `Delete`

cluster-config.local-storage.default

Type: `bool` **Required:** `No`

Determines if the storage class should be set as default. If omitted defaults to `true`

cluster-config.gateway

Type: `object` **Required:** `No`

Configuration options for the gateway feature

cluster-config.gateway.enabled

Type: `bool` **Required:** `No`

Determines if the feature should be enabled. If omitted defaults to `true`

cluster-config.cloud-provider

Type: `string` **Required:** `No` **Possible Values:** `external`

Sets the cloud provider to be used by the cluster.

When this is set as `external`, node will wait for an external cloud provider to do cloud specific setup and finish node initialization.

control-plane-taints

Type: `list[string]` **Required:** `No`

List of taints to be applied to control plane nodes.

pod-cidr

Type: `string` **Required:** `No`

The CIDR to be used for assigning pod addresses. If omitted defaults to `10.1.0.0/16`

service-cidr

Type: `string` **Required:** `No`

The CIDR to be used for assigning service addresses. If omitted defaults to `10.152.183.0/24`

disable-rbac

Type: `bool` **Required:** `No`

Determines if RBAC should be disabled. If omitted defaults to `false`

secure-port

Type: `int` **Required:** `No`

The port number for kube-apiserver to use. If omitted defaults to `6443`

k8s-dqlite-port

Type: `int` **Required:** `No`

The port number for k8s-dqlite to use. If omitted defaults to `9000`

datastore-type

Type: `string` **Required:** `No` **Possible Values:** `k8s-dqlite | external`

The type of datastore to be used. If omitted defaults to `k8s-dqlite`

Can be used to point to an external datastore like etcd.

datastore-servers

Type: `list[string]` **Required:** `No`

The server addresses to be used when `datastore-type` is set to `external`.

datastore-ca-crt

Type: `string` **Required:** `No`

The CA certificate to be used when communicating with the external datastore.

datastore-client-crt

Type: `string` **Required:** `No`

The client certificate to be used when communicating with the external datastore.

datastore-client-key

Type: `string` **Required:** `No`

The client key to be used when communicating with the external datastore.

extra-sans

Type: `list[string]` **Required:** `No`

List of extra SANs to be added to certificates.

ca-crt

Type: `string` **Required:** `No`

The CA certificate to be used for Kubernetes services. If omitted defaults to an auto generated certificate.

ca-key

Type: `string` **Required:** `No`

The CA key to be used for Kubernetes services. If omitted defaults to an auto generated key.

front-proxy-ca-crt

Type: `string` **Required:** `No`

The CA certificate to be used for the front proxy. If omitted defaults to an auto generated certificate.

front-proxy-ca-key

Type: `string` **Required:** `No`

The CA key to be used for the front proxy. If omitted defaults to an auto generated key.

front-proxy-client-crt

Type: `string` **Required:** `No`

The client certificate to be used for the front proxy. If omitted defaults to an auto generated certificate.

front-proxy-client-key

Type: `string` **Required:** `No`

The client key to be used for the front proxy. If omitted defaults to an auto generated key.

apiserver-kubelet-client-crt

Type: `string` **Required:** `No`

The client certificate to be used by kubelet for communicating with the kube-apiserver. If omitted defaults to an auto generated certificate.

apiserver-kubelet-client-key

Type: `string` **Required:** `No`

The client key to be used by kubelet for communicating with the kube-apiserver. If omitted defaults to an auto generated key.

service-account-key

Type: `string` **Required:** `No`

The key to be used by the default service account. If omitted defaults to an auto generated key.

apiserver-crt

Type: `string` **Required:** `No`

The certificate to be used for the kube-apiserver. If omitted defaults to an auto generated certificate.

apiserver-key

Type: `string` **Required:** `No`

The key to be used for the kube-apiserver. If omitted defaults to an auto generated key.

kubelet-crt

Type: `string` **Required:** `No`

The certificate to be used for the kubelet. If omitted defaults to an auto generated certificate.

kubelet-key

Type: `string` **Required:** `No`

The key to be used for the kubelet. If omitted defaults to an auto generated key.

5.3.2 Example

The following example configures and enables certain features, sets an external cloud provider, marks the control plane nodes as unschedulable, changes the pod and service CIDRs from the defaults and adds an extra SAN to the generated certificates.

```
cluster-config:
  network:
    enabled: true
  dns:
    enabled: true
    cluster-domain: cluster.local
  ingress:
    enabled: true
  load-balancer:
    enabled: true
    cidrs:
    - 10.0.0.0/24
    - 10.1.0.10-10.1.0.20
    l2-mode: true
  local-storage:
    enabled: true
    local-path: /storage/path
    default: false
  gateway:
    enabled: true
  metrics-server:
    enabled: true
  cloud-provider: external
control-plane-taints:
- node-role.kubernetes.io/control-plane:NoSchedule
pod-cidr: 10.100.0.0/16
service-cidr: 10.200.0.0/16
disable-rbac: false
secure-port: 6443
k8s-dqlite-port: 9090
datastore-type: k8s-dqlite
extra-sans:
- custom.kubernetes
```

5.4 Proxy environment variables

Canonical Kubernetes uses the standard system-wide environment variables to control access through proxies.

On Ubuntu and other Linux operating systems, proxies are configured through system-wide environment variables defined in the `/etc/environment` file.

- **HTTPS_PROXY**
- **HTTP_PROXY**
- **NO_PROXY**
- **https_proxy**

- **http_proxy**
- **no_proxy**

5.4.1 No-proxy CIDRS

When configuring proxies, it is important to note that there are always some CIDRs which need to be excluded and added to the `no-proxy` lists. For Canonical Kubernetes these are:

- The range used by Kubernetes services (defaults to **10.152.183.0/24**)
- The range used by the Kubernetes pods (defaults to **10.1.0.0/16**)

And it is also important to exclude the local network to maintain access to any local traffic.

5.4.2 Configuring

For the `k8s` snap, proxy configuration is controlled by editing the `etc/environment` file mentioned above. There is an example in the *How to guide for configuring proxies for the k8s snap*.

For charms deployed by Juju, proxies are managed by configuring the model. See the *How to guide for configuring proxies for k8s charms* for an example of how to set these.

5.5 Troubleshooting

This page provides techniques for troubleshooting common Canonical Kubernetes issues.

5.5.1 Kubectl error: “dial tcp 127.0.0.1:6443: connect: connection refused”

Problem

The kubeconfig file generated by the `k8s kubectl` CLI can not be used to access the cluster from an external machine. The following error is seen when running `kubectl` with the invalid kubeconfig:

```
...
E0412 08:36:06.404499  517166 memcache.go:265] couldn't get current server API group␣
↪list: Get "https://127.0.0.1:6443/api?timeout=32s": dial tcp 127.0.0.1:6443: connect:␣
↪connection refused
The connection to the server 127.0.0.1:6443 was refused - did you specify the right host␣
↪or port?
```

Explanation

A common technique for viewing a cluster kubeconfig file is by using the `kubectl config view` command.

The `k8s kubectl` command invokes an integrated `kubectl` client. Thus `k8s kubectl config view` will output a seemingly valid kubeconfig file. However, this will only be valid on cluster nodes where control plane services are available on localhost endpoints.

Solution

Use `k8s config` instead of `k8s kubectl config` to generate a kubeconfig file that is valid for use on external machines.

5.6 Architecture

A system architecture document is the starting point for many interested participants in a project, whether you intend contributing or simply want to understand how the software is structured. This documentation lays out the current design of Canonical Kubernetes, following the C4 model.

5.6.1 System context

This overview of Canonical Kubernetes demonstrates the interactions of Kubernetes with users and with other systems.

Two actors interact with the Kubernetes snap:

- **K8s admin**: The administrator of the cluster interacts directly with the Kubernetes API server. Out of the box our K8s distribution offers admin access to the cluster. That initial user is able to configure the cluster to match their needs and of course create other users that may or may not have admin privileges. The K8s admin is also able to maintain workloads running in the cluster.
- **K8s user**: A user consuming the workloads hosted in the cluster. Users do not have access to the Kubernetes API server. They need to access the cluster through the options (nodeport, ingress, load-balancer) offered by the administrator who deployed the workload they are interested in.

There are non-human users of the K8s snap, for example the `k8s-operator charm`. The K8s charm needs to drive the Kubernetes cluster and to orchestrate the multi-node clustering operations.

A set of external systems need to be easily integrated with our K8s distribution. We have identified the following:

- **Load Balancer**: Although the K8s snap distribution comes with a load balancer we expect the end customer environment to have a load balancer and thus we need to integrate with it.
- **Storage**: Kubernetes typically expects storage to be external to the cluster. The K8s snap comes with a local storage option but we still need to offer proper integration with any storage solution.
- **Identity management**: Out of the box the K8s snap offers credentials for an admin user. The admin user can complete the integration with any identity management system available or do user management manually.
- **External datastore**: By default, Kubernetes uses etcd to keep track of state. Our K8s snap comes with `dqlite` as its datastore. We should however be able to use any end client owned datastore installation. That should include an external `postgresql` or `etcd`.

5.6.2 The k8s snap

Looking more closely at what is contained within the K8s snap itself:

The `k8s` snap distribution includes the following:

- **Kubectl**: through which users and other systems interact with Kubernetes and drive the cluster operations.
- **K8s services**: These are all the Kubernetes services as well as core workloads built from upstream and shipped in the snap.

- State is backed up by **dqlite** by default, which keeps that state of the Kubernetes cluster as well as the state we maintain for the needs of the cluster operations. The cluster state may optionally be stored in a different, external datastore.
- **Runtime**: `containerd` and `runc` are the shipped container runtimes.
- **K8sd**: which implements the operations logic and exposes that functionality via CLIs and APIs.

5.6.3 K8sd

K8sd is the component that implements and exposes the operations functionality needed for managing the Kubernetes cluster.

At the core of the `k8sd` functionality we have the cluster manager that is responsible for configuring the services, workload and features we deem important for a Kubernetes cluster. Namely:

- Kubernetes systemd services
- DNS
- CNI
- ingress
- gateway API
- load-balancer
- local-storage
- metrics-server

The cluster manager is also responsible for implementing the formation of the cluster. This includes operations such as joining/removing nodes into the cluster and reporting status.

This functionality is exposed via the following interfaces:

- The **CLI**: The CLI is available to only the root user on the K8s snap and all CLI commands are mapped to respective REST calls.
- The **API**: The API over HTTP serves the CLI and is also used to programmatically drive the Kubernetes cluster.

5.6.4 Canonical K8s charms

Canonical `k8s` Charms encompass two primary components: the `k8s` charm and the `k8s-worker` charm.

Charms are instantiated on a machine as a Juju unit, and a collection of units constitutes an application. Both `k8s` and `k8s-worker` units are responsible for installing and managing its machine's `k8s` snap, however the charm type determines the node's role in the Kubernetes cluster. The `k8s` charm manages `control-plane` nodes, whereas the `k8s-worker` charm manages Kubernetes `worker` nodes. The administrator manages the cluster via the `juju` client, directing the `juju` controller to reach the model's eventually consistent state. For more detail on Juju's concepts, see the Juju docs.

The administrator may choose any supported cloud-types (Openstack, MAAS, AWS, GCP, Azure…) on which to manage the machines making up the Kubernetes cluster. Juju selects a single leader unit per application to act as a centralised figure with the model. The `k8s` leader oversees Kubernetes bootstrapping and enlistment of new nodes. Follower `k8s` units will join the cluster using secrets shared through relation data from the leader. The entire lifecycle of the deployment is orchestrated by the `k8s` charm, with tokens and cluster-related information being exchanged through Juju relation data.

Furthermore, the `k8s-worker` unit functions exclusively as a worker within the cluster, establishing a relation with the `k8s` leader unit and requesting tokens and cluster-related information through relation data. The `k8s` leader is responsible for issuing these tokens and revoking them if a unit administratively departs the cluster.

The `k8s` charm also supports the integration of other compatible charms, enabling integrations such as connectivity with an external `etcd` datastore and the sharing of observability data with the `Canonical Observability Stack (COS)`. This modular and integrated approach facilitates a robust and flexible Canonical Kubernetes deployment managed through Juju.

5.7 Welcome to the Canonical Kubernetes community

This rapidly growing community is a diverse bunch of people - developers, Kubernetes admins, inventors, researchers, students… and we all share the joy of a reliable, flexible, secure, timely version of upstream Kubernetes. The team recognise the important role each and every user plays in the success of the project as a whole and how valuable your contributions are.

5.7.1 Do you have questions?

Do you have questions about Canonical Kubernetes? Perhaps you want some ideas on how to best achieve a certain goal or maybe some aspect of your Kubernetes doesn't behave the way you expect. Perhaps you'd just like some advice from more experienced users. There are a number of ways to get in touch:

- Using the Kubernetes slack: find us in the #canonical-kubernetes channel
- In the public Matrix room
- On the Ubuntu Discourse

For more formal support, please see the support options available to you on the Ubuntu website.

Our commitment to you - we may not always be able to answer your questions, but we promise to respond within three working days.

5.7.2 Found a bug?

You can always track what is going on with development by watching our GitHub repository. This is also the best place to file a bug if you find one, or of course you are also welcome to contribute to the code.

Our commitment to you - we monitor the issues on GitHub regularly and we aim to triage all bug reports within three working days.

5.7.3 Contributing to the code?

Canonical Kubernetes is proudly open source, published under the GPLv3 license. We welcome and encourage contributions to the code. Please see the *Developer guide* for more information on contributing.

Our commitment to you - we closely follow activity on the source repository. We aim to respond to any PRs within three working days.

5.7.4 Contributing to docs?

Our documentation is extremely important to us and is actively maintained by the entire team. That doesn't mean that it can't be improved though. Every page in the documentation has an "Edit this page" link in the top right which takes you to GitHub to make small changes. For larger contributions, please see the *Contributing guide*.

Our commitment to you: Comments are usually read daily and we are really grateful for docs improvements.

5.7.5 Code of conduct

Building a fair, open and inclusive community is important to us. We think adopting a code of conduct is a sensible way to make sure that everybody participating understands what the expectations and obligations are. The team adopts the Ubuntu Code of Conduct 2.0, and we use these as the guidelines for participation.

5.8 Roadmap

The Canonical Kubernetes team enthusiastically supports the idea of a public roadmap, letting everyone know the headline features we are working on and the future direction and priorities of the project.

Our roadmap matches the cadence of the Ubuntu release cycle, so `24.10` is the same as the release date for Ubuntu 24.10. This does not precisely map to the release cycle of Kubernetes versions, so please consult the *release notes* for specifics of whatfeatures have been delivered.

Table 1: Canonical Kubernetes public roadmap

24.04	24.10
k8s snap based on the upcoming Ubuntu 24.04 LTS release	Security features
k8s charm	Compliance

5.9 Other documentation types

Alternatively, the *Tutorials section* contains step-by-step tutorials to help guide you through exploring and using Canonical Kubernetes.

If you have a specific goal our *How-to guides* have more in-depth detail than tutorials and can be applied to a broader set of applications. They'll help you achieve an end-result but may require you to understand and adapt the steps to fit your specific requirements.

Finally, for a better understanding of how Canonical Kubernetes works and related topics such as security, our *Explanation section* helps you to expand your knowledge and get the most out of Kubernetes.

CHAPTER

SIX

CANONICAL KUBERNETES CHARM DOCUMENTATION

The Canonical Kubernetes charm, `k8s`, is an operator: software which wraps an application and contains all of the instructions necessary for deploying, configuring, scaling, integrating the application on any cloud supported by Juju.

The `k8s` charm takes care of installing and configuring the *k8s snap package* on cloud instances managed by Juju. Operating Kubernetes through this charm makes it significantly easier to manage at scale, on remote cloud instances and also to integrate other operators to enhance or customise your Kubernetes deployment. You can find out more about Canonical Kubernetes on this *overview page* or see a more detailed explanation in our *architecture documentation*.

6.1 In this documentation

Tutorial **Start here!** A hands-on introduction to Canonical K8s 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

6.2 Project and community

Canonical Kubernetes is a member of the Ubuntu family. It's an open source project which welcomes community involvement, contributions, suggestions, fixes and constructive feedback.

- Our Code of Conduct
- Our *community*
- How to *contribute*
- Our development *roadmap*

CHAPTER

SEVEN

TUTORIALS

This section contains a step-by-step guide to help you start exploring how to install and use Canonical Kubernetes.

7.1 Getting started

The Canonical Kubernetes `k8s` charm takes care of installing and configuring Kubernetes on cloud instances managed by Juju. Operating Kubernetes through this charm makes it significantly easier to manage at scale, on remote cloud instances and also to integrate other operators to enhance or customise your Kubernetes deployment. This tutorial will take you through installing Kubernetes and some common first steps.

7.1.1 What you will learn

- How to install Canonical Kubernetes
- Making a cluster
- Deploying extra workers
- Using Kubectl

7.1.2 What you will need

- Ubuntu 22.04 LTS or 20.04 LTS
- The Juju client
- Access to a Juju-supported cloud for creating the required instances
- Kubectl for interacting with the cluster (installation instructions are included in this tutorial)

7.1.3 1. Get prepared

Deploying charms with Juju requires a substrate or backing cloud to actually run the instances. If you are unfamiliar with Juju, it would be useful to run through the Juju tutorial first, and ensure you have a usable controller to deploy with.

Before installing anything, we should first check what versions of the charm are available. Charms are published to 'channels' which reflect both a specific release version and the maturity or stability of that code. Sometimes we may wish to run on the latest stable version, sometimes the goal is to test out upcoming features or test migration to a new version. Channels are covered in more detail in *the channel explanation page* if you want to learn more. The currently available versions of the charm can be discovered by running:

```
juju info k8s
```

or

```
juju info k8s-worker
```

There are two distinct charms - one includes control-plane services for administering the cluster, the other omits these for nodes which are to be deployed purely for workloads. Both are published simultaneously from the same source so the available channels should match. Running the commands will output basic information about the charm, including a list of the available channels at the end.

Charm deployments default to "latest/stable", but if you want to chose a specific version it can be indicated when deploying with the `--channel=` argument, for example `--channel=latest/edge`.

7.1.4 2. Deploy the K8s charm

To make sure that Juju creates an instance which has enough resources to actually run Kubernetes, we will make use of 'constraints'. These specify the minimums required. For the Kubernetes control plane (`k8s` charm), the recommendation is two CPU cores, 16GB of memory and 40GB of disk space. Now we can go ahead and create a cluster:

```
juju deploy k8s --channel=latest/edge --constraints='cores=2 mem=16G root-disk=40G'
```

At this point Juju will fetch the charm from Charmhub, create a new instance according to your specification and configure and install the Kubernetes components (i.e. the `k8s` snap). This may take a few minutes depending on your cloud. You can monitor progress by watching the Juju status output:

```
juju status --watch 2s
```

When the status reports that K8s is "idle/ready" you have successfully deployed a Canonical Kubernetes control-plane using Juju.

Note: For High Availability you will need at least three units of the k8s charm. Scaling the deployment is covered below.

7.1.5 3. Deploy a worker

Before we start doing things in Kubernetes, we should consider adding a worker. The K8s worker is an additional node for the cluster which focuses on running workloads without running any control-plane services. This means it needs a connection to a control-plane node to tell it what to do, but it also means more of its resources are available for running workloads. We can deploy a worker node in a similar way to the original K8s node:

```
juju deploy k8s-worker --channel=latest/edge --constraints='cores=2 mem=16G root-disk=40G
↪'
```

Once again, this will take a few minutes. In this case though, the `k8s-worker` application won't settle into a 'Ready' status, because it requires a connection to the control plane. This is handled in Juju by integrating the charms so they can communicate using a standard interface. The charm info we fetched earlier also includes a list of the relations possible, and from this we can see that the k8s-worker requires "cluster: k8s-cluster".

To connect these charms and effectively add the worker to our cluster, we use the 'integrate' command, adding the interface we wish to connect

```
juju integrate k8s k8s-worker:cluster
```

After a short time, the worker node will share information with the control plane and be joined to the cluster.

7.1.6 4. Scale the cluster (Optional)

If one worker doesn't seem like enough, we can easily add more:

```
juju add-unit k8s-worker -n 1
```

This will create an additional instance running the k8s-worker charm. Juju manages all instances of the same application together, so there is no need to add the integration again. If you check the Juju status, you should see that a new unit is created, and now you have `k8s-worker/0` and `k8s-worker/1`

7.1.7 5. Use Kubectl

Kubectl is the standard upstream tool for interacting with a Kubernetes cluster. This is the command that can be used to inspect and manage your cluster.

If you don't already have `kubectl`, it can be installed from a snap:

```
sudo snap install kubectl --classic
```

If you have just installed it, you should also create a file to contain the configuration:

```
mkdir ~/.kube
```

To fetch the configuration information from the cluster we can run:

```
juju run k8s/0 get-kubeconfig
```

The Juju action is a piece of code which runs on a unit to perform a specific task. In this case it collects the cluster information - the YAML formatted details of the cluster and the keys required to connect to it.

> **Warning:** If you already have Kubectl and are using it to manage other clusters, you should edit the relevant parts of the cluster yaml output and append them to your current kubeconfig file.

We can use pipe to append your cluster's kubeconfig information directly to a config file which will just require a bit of editing:

```
juju run k8s/0 get-kubeconfig >> ~/.kube/config
```

The output includes the root of the YAML, `kubeconfig: |`, so we can just use an editor to remove that line:

```
nano ~/.kube/config
```

Please use the editor of your choice to delete the first line and save the file.

Alternatively, if you are a `yq` user, the same can be achieved with:

```
juju run k8s/0 get-kubeconfig | yq '.kubeconfig' -r >> ~/.kube/config
```

You can now confirm Kubectl can read the kubeconfig file:

```
kubectl config show
```

. . . which should output something like this:

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://10.158.52.236:6443
  name: k8s
contexts:
- context:
    cluster: k8s
    user: k8s-user
  name: k8s
current-context: k8s
kind: Config
preferences: {}
users:
- name: k8s-user
  user:
    token: REDACTED
```

You can then further confirm that it is possible to inspect the cluster by running a simple command such as :

```
kubectl get pods -A
```

This should return some pods, confirming the command can reach the cluster:

```
NAMESPACE     NAME                                READY   STATUS    RESTARTS   AGE
kube-system   cilium-4m5xj                        1/1     Running   0          35m
kube-system   cilium-operator-5ff9ddcfdb-b6qxm    1/1     Running   0          35m
kube-system   coredns-7d4dffcffd-tvs6v            1/1     Running   0          35m
kube-system   metrics-server-6f66c6cc48-wdxxk     1/1     Running   0          35m
```

7.1.8 Next steps

Congratulations - you now have a functional Kubernetes cluster! In the near future more charms are on the way to simplify usage and extend the base functionality of Canonical Kubernetes. Bookmark the *releases page* to keep informed of updates.

7.2 Other documentation types

If you have a specific goal our *How-to guides* have more in-depth detail than tutorials and can be applied to a broader set of applications. They'll help you achieve an end-result but may require you to understand and adapt the steps to fit your specific requirements.

For a better understanding of how Canonical Kubernetes works and related topics such as security, our *Explanation section* helps you to expand your knowledge and get the most out of Kubernetes.

Finally, our *Reference section* is for when you need to check specific details or information such as the command reference or release notes.

CHAPTER

EIGHT

HOW-TO GUIDES

If you have a specific goal, but are already familiar with Kubernetes, our How-to guides are more specific and contain less background information. They'll help you achieve an end result but may require you to understand and adapt the steps to fit your specific requirements.

8.1 Install Canonical Kubernetes from a charm

Canonical Kubernetes is packaged as a charm, available from Charmhub for all supported platforms.

8.1.1 What you'll need

This guide assumes the following:

- The rest of this page assumes you already have Juju installed and have added credentials for a cloud and bootstrapped a controller.
- If you still need to do this, please take a look at the quickstart instructions, or, for custom clouds (OpenStack, MAAS), please consult the Juju documentation.

Note: If you cannot meet these requirements, please see the *Installing* page for alternative options.

8.1.2 Check available channels (optional)

It is a good idea to check the available channels before installing the charm. Run the command:

```
juju info k8s
juju info k8s-worker
```

…which will output a list of currently available channels. See the *channels page* for an explanation of the different types of channel.

8.1.3 Deploying the charm

The charm can be installed with the `juju` command:

```
juju deploy k8s --channel=latest/edge
```

Note: The `latest/edge` channel is always under active development. This is where you will find the latest features but you may also experience instability.

8.1.4 Bootstrap the cluster

Installing the k8s charm sets up all the parts required to run Kubernetes. One may watch it progress using juju status:

```
juju status --watch 1s
```

This command will output a message confirming the charm is deployed and the cluster is bootstrapped. It is recommended to ensure that the cluster initialises properly and is running with no issues.

Once the unit is active/idle, You'll know the cluster is installed.

8.1.5 Expanding the cluster

At this point, you should have one control-plane node. To expand your cluster, add more units with the following command

```
juju add-unit k8s -n 2
```

This will create 2 more control-plane units clustered with the first.

Use `juju status` to watch these units approach active/idle

8.1.6 Adding Workers

In many cases, it is desirable to have additional 'worker only' units in the cluster. Rather than adding more control-plane units, we'll deploy the `k8s-worker` charm. After deployment, integrate these new nodes with control-plane units so they join the cluster.

```
juju deploy k8s-worker --channel=latest/edge -n 2
juju integrate k8s k8s-worker:cluster
```

Use `juju status` to watch these units approach the active/idle state.

8.2 How to integrate Canonical Kubernetes with etcd

Integrating **etcd** with your Canonical Kubernetes deployment provides a robust, distributed key-value store that is essential for storing critical data needed for Kubernetes' clustering operations. This guide will walk you through the process of deploying Canonical Kubernetes with an external etcd cluster.

8.2.1 What you will need

- A Juju controller with access to a cloud environment (see the Juju setup guide for more information).

Warning: Once you deploy your Canonical Kubernetes cluster with a particular datastore, you cannot switch to a different datastore post-deployment. Planning for your datastore needs ahead of time is crucial, particularly if you opt for an external datastore like **etcd**.

8.2.2 Preparing the Deployment

1. **Creating the Deployment Model**: Begin by creating a Juju model specifically for your Canonical Kubernetes cluster deployment.

```
juju add-model my-cluster
```

2. **Deploying Certificate Authority**: etcd requires a secure means of communication between its components. Therefore, we require a certificate authority such as EasyRSA or Vault. Check the respective charm documentation for detailed instructions on how to deploy a certificate authority. In this guide, we will be using EasyRSA.

```
juju deploy easyrsa
```

8.2.3 Deploying etcd

- **Single Node Deployment**:
  - To deploy a basic etcd instance on a single node, use the command:

    ```
    juju deploy etcd
    ```

    This setup is straightforward but not recommended for production environments due to a lack of high availability.
- **High Availability Setup**:
  - For environments where high availability is crucial, deploy etcd across at least three nodes:

    ```
    juju deploy etcd -n 3
    ```

    This ensures that your etcd cluster remains available even if one node fails.

8.2.4 Integrating etcd with EasyRSA

Now you have to integrate etcd with your certificate authority; this will issue the required certificates for secure communication between etcd and your Canonical Kubernetes cluster:

```
juju integrate etcd easyrsa
```

8.2.5 Deploying Canonical Kubernetes

Deploy the control plane units of Canonical Kubernetes with the command:

```
juju deploy k8s --config datastore=etcd -n 3
```

This command deploys 3 units of the Canonical Kubernetes control plane (`k8s`) and configures them to use **etcd** as the backing datastore, ensuring high availability.

Important: Remember to run `juju expose k8s`. This will open the required ports to reach your cluster from outside.

8.2.6 Integrating Canonical Kubernetes with etcd

Now that we have both the etcd datastore deployed alongside our Canonical Kubernetes cluster, it is time to integrate our cluster with our etcd datastore.

```
juju integrate k8s etcd
```

This step integrates the k8s charm (Control Plane units) with the etcd hosts, allowing the Kubernetes cluster to use the etcd units as an external datastore.

8.2.7 Final Steps

Verify the Deployment: After completing the deployment, it's essential to verify that all components are functioning correctly. Use the `juju status` command to inspect the current status of your cluster.

```
Model       Controller  Cloud/Region   Version  SLA          Timestamp
my-cluster  canonicaws  aws/us-east-1  3.4.2    unsupported  16:02:18-05:00

App      Version  Status  Scale  Charm    Channel        Rev  Exposed  Message
easyrsa  3.0.1    active      1  easyrsa  latest/stable   58  no       Certificate␣
↪Authority connected.
etcd     3.4.22   active      3  etcd     latest/stable  760  no       Healthy with 3␣
↪known peers
k8s      1.29.4   active      3  k8s      latest/edge     33  yes      Ready

Unit        Workload  Agent  Machine  Public address  Ports     Message
easyrsa/0*  active    idle   0        35.172.230.66             Certificate Authority␣
↪connected.
etcd/0*     active    idle   1        34.204.173.161  2379/tcp  Healthy with 3 known␣
↪peers
```

(continues on next page)

(continued from previous page)

```
etcd/1      active    idle   2        54.225.4.183    2379/tcp  Healthy with 3 known␣
↪peers
etcd/2      active    idle   3        3.208.15.61     2379/tcp  Healthy with 3 known␣
↪peers
k8s/0       active    idle   4        54.89.153.117   6443/tcp  Ready
k8s/1*      active    idle   5        3.238.230.3     6443/tcp  Ready
k8s/2       active    idle   6        34.229.202.243  6443/tcp  Ready

Machine  State    Address         Inst id              Base          AZ          Message
0        started  35.172.230.66   i-0b6fc845c28864913  ubuntu@22.04  us-east-1f  running
1        started  34.204.173.161  i-05439714c88bea35f  ubuntu@22.04  us-east-1f  running
2        started  54.225.4.183    i-07ecf97ed29860334  ubuntu@22.04  us-east-1c  running
3        started  3.208.15.61     i-0be91170809d7dccc  ubuntu@22.04  us-east-1b  running
4        started  54.89.153.117   i-07906e76071b69721  ubuntu@22.04  us-east-1c  running
5        started  3.238.230.3     i-0773583e7a5fbf07e  ubuntu@22.04  us-east-1f  running
6        started  34.229.202.243  i-0f03b9833a338380c  ubuntu@22.04  us-east-1b  running
```

8.3 Configuring proxy settings for K8s

Canonical Kubernetes packages a number of utilities (eg curl, helm) which need to fetch resources they expect to find on the internet. In a constrained network environment, such access is usually controlled through proxies.

8.3.1 Adding proxy configuration for the k8s charms

For the charm deployments of Canonical Kubernetes, Juju manages proxy configuration through the Juju model.

For example, assume we have a proxy running at `http://squid.internal:3128` and we are using the networks `10.0.0.0/8`,`192.168.0.0/16` and `172.16.0.0/12`. In this case we would configure the model in which the charms are to run with Juju:

```
juju model-config \
    juju-http-proxy=http://squid.internal:3128 \
    juju-https-proxy=http://squid.internal:3128 \
    juju-no-proxy=10.0.8.0/24,192.168.0.0/16,127.0.0.1,10.152.183.0/24
```

Note: The **10.152.183.0/24** CIDR needs to be covered in the juju-no-proxy list as it is the Kubernetes service CIDR. Without this any pods will not be able to reach the cluster's kubernetes-api. You should also exclude the range used by pods (which defaults to **10.1.0.0/16**) and any required local networks.

8.4 Integrating with COS Lite

It is often advisable to have a monitoring solution which will run whether the cluster itself is running or not. It may also be useful to integrate monitoring into existing setups.

To make monitoring your cluster a delightful experience, Canonical provides first-class integration between Canonical Kubernetes and COS Lite (Canonical Observability Stack). This guide will help you integrate a COS Lite deployment with a Canonical Kubernetes deployment.

This document assumes you have a controller with an installation of Canonical Kubernetes. If you have not yet installed Canonical Kubernetes, please see *"Installing Canonical Kubernetes"*.

8.4.1 Preparing a platform for COS Lite

If you are unfamiliar with Juju models, the documentation can be found here. In this section, we'll be adding a new model to keep observability separate from the Kubernetes model.

First, create a MicroK8s model to act as a deployment cloud for COS Lite:

```
juju add-model --config logging-config='<root>=DEBUG' microk8s-ubuntu
```

We also set the logging level to DEBUG so that helpful debug information is shown when you use `juju debug-log` (see juju debug-log).

Use the Ubuntu charm to deploy an application named "microk8s":

```
juju deploy ubuntu microk8s --series=focal --constraints="mem=8G cores=4 root-disk=30G"
```

Deploy MicroK8s on Ubuntu by accessing the unit you created at the last step with `juju ssh microk8s/0` and following the Install Microk8s guide for configuration.

Note: Make sure to enable the hostpath-storage and MetalLB addons for Microk8s.

Export the Microk8s kubeconfig file to your current directory after configuration:

```
juju ssh microk8s/0 -- microk8s config > microk8s-config.yaml
```

Register MicroK8s as a Juju cloud using add-k8s (see "juju add-k8s" for details on the add-k8s command):

```
KUBECONFIG=microk8s-config.yaml juju add-k8s microk8s-cloud
```

8.4.2 Deploying COS Lite on the Microk8s cloud

On the Microk8s cloud, create a new model and deploy the `cos-lite` bundle:

```
juju add-model cos-lite microk8s-cloud
juju deploy cos-lite
```

Make COS Lite's endpoints available for cross-model integration:

```
juju offer grafana:grafana-dashboard
juju offer prometheus:receive-remote-write
```

Use `juju status --relations` to verify that both `grafana` and `prometheus` offerings are listed.

At this point, you've established a MicroK8s model on Ubuntu and incorporated it into Juju as a Kubernetes cloud. You then used this cloud as a substrate for the COS Lite deployment. You therefore have 2 models on the same controller.

8.4.3 Integrating COS Lite with Canonical Kubernetes

Switch to your Canonical Kubernetes model (if you forgot the name of your model, you can run `juju models` to see a list of available models):

```
juju switch <canonical-kubernetes-model>
```

Consume the COS Lite endpoints:

```
juju consume cos-lite.grafana
juju consume cos-lite.prometheus
```

Deploy the grafana-agent:

```
juju deploy grafana-agent
```

Relate `grafana-agent` to `k8s`:

```
juju integrate grafana-agent:cos-agent k8s:cos-agent
```

Relate `grafana-agent` to the COS Lite offered interfaces:

```
juju integrate grafana-agent grafana
juju integrate grafana-agent prometheus
```

Get the credentials and login URL for Grafana:

```
juju run grafana/0 get-admin-password -m cos-lite
```

The above command will output:

```
admin-password: b9OhxF5ndUDO
url: http://10.246.154.87/cos-lite-grafana
```

The username for this credential is `admin`.

You've successfully gained access to a comprehensive observability stack. Visit the URL and use the credentials to log in.

Once you feel ready to dive deeper into your shiny new observability platform, you can head over to the COS Lite documentation.

8.5 How to contribute to Canonical Kubernetes

Canonical Kubernetes is proudly open source, published under the GPLv3 license. We welcome and encourage contributions to the code and the documentation. See the *community page* for ways to get in touch and provide feedback.

8.5.1 Contribute to the code

Canonical Kubernetes is shipped as a snap package. To contribute to the code, you should first make sure you can build and test the snap locally.

Build the snap

To build the snap locally, you will need the following:

- The latest, snap-based version of LXD (see the install guide here)
- The Snapcraft build tool, for building the snap (see the Snapcraft documentation).

Clone the GitHub repository for the k8s snap and then open a terminal in that directory. Run the command:

```
snapcraft --use-lxd
```

This will launch an LXD container and use it to build a version of the snap. This will take some time as the build process fetches dependencies, stages the 'parts' of the snap and creates the snap package itself. The snap itself will be fetched from the build environment and placed in the local project directory. Note that the LXD container used for building will be stopped, but not deleted. This is in case there were any errors or artefacts you may wish to inspect.

Install the snap

The snap can then be installed locally by using the '–dangerous' option. This is a safeguard to make sure the user is aware that the snap is not signed by the snap store, and is not confined:

```
sudo snap install k8s_v1.29.2_multi.snap --dangerous --classic
```

Note: You will not be able to install this snap if there is already a k8s snap installed on your system.

Once you have verified the current snap build works, it can be removed with:

```
sudo snap remove k8s --purge
```

The `purge` option is recommended when iterating over code changes, as it also removes all the installed artefacts which may be associated with the snap.

Now you can iterate over changes to the snap, rebuild and test.

As noted previously, the LXD container used for building is not removed and will be reused by subsequent build instructions. When you are satisfied it is no longer needed, this container can be removed:

```
lxc delete snapcraft-k8s
```

Contribute changes

We welcome any improvements and bug-fixes to the Canonical Kubernetes code. Once you have tested your changes, please make a pull request on the code repository and we will review it as soon as possible.

8.5.2 Contribute to the documentation

Our aim is to provide easy-to-understand documentation on all aspects of Canonical Kubernetes, so we greatly appreciate your feedback and contributions. See our *community page* for ways of getting in touch.

The source of the documentation and the system used to build it are included in the main repository for the Canonical Kubernetes snap.

Documentation framework

This documentation has adopted the Diátaxis framework. You can read more about it on the Diátaxis website. In essence though, this guides the way we categorise and write our documentation. You can see there are four main categories of documentation:

- **Tutorials** for guided walkthroughs
- **How to** pages for specific tasks and goals
- **Explanation** pages which give background reasons and, well, explanations
- **Reference**, where you will find the commands, the roadmap, etc.

Every page of documentation should fit into one of those categories. If it doesn't you may consider if it is actually two pages (e.g. a How to *and* an explanation).

Small changes

If you are simply correcting a typo or updating a link, you can follow the 'Edit this page on GitHub' link on any page and it will take you to the online editor to make your change. You will still need to raise a pull request and explain your change to get it reviewed.

Myst, Markdown and Sphinx

We use the Sphinx documentation tools to actually build the documentation. You will find all the Sphinx tooling in the `docs/tools` directory.

Although Sphinx is normally associated with the `ReSTructured text` format, we write all our documentation in Markdown to make it easier for humans to work with. There are a few extra things that come with this - certain features need to be specially marked up (e.g. admonitions) to be processed properly. There is a guide to using `Myst` (which is a Markdown extension for Sphinx) directives and formatting contained in the _parts directory of the docs.

Local testing

To test your changes locally, you can build a local version of the documentation. Open a terminal and go to the `/docs/tools` directory. From there you can run the command:

```
make run
```

This will create a local environment, install all the dependencies and build the docs. The output will then be served locally - check the output for the URL. Using the `run` option means that the docs will automatically be regenerated when you change any of the source files too (though remember to press `F5` in your browser to reload the page without caching)!

8.6 Other documentation types

Our Reference section is for when you need to check specific details or information such as the command reference or release notes.

Alternatively, the *Tutorials section* contains step-by-step tutorials to help guide you through exploring and using Canonical Kubernetes.

For a better understanding of how Canonical Kubernetes works and related topics such as security, our *Explanation section* helps you expand your knowledge and get the most out of Kubernetes.

Finally, our *Reference section* is for when you need to check specific details or information such as the command reference or release notes.

CHAPTER

NINE

EXPLANATION

For a better understanding of how Canonical Kubernetes works and related topics such as security, these pages will help expand your knowledge and help you get the most out of Kubernetes.

9.1 What is Canonical Kubernetes?

At its core, Canonical Kubernetes is a full implementation of upstream Kubernetes delivered in a compact, secure, reliable snap package. As the upstream Kubernetes services are not all that is required for a fully functional cluster, additional services and features are built in. You can deploy the snap and have a single-node cluster up and running in minutes.

9.1.1 Why a snap?

Snaps are self-contained, simple to install, secure, cross-platform, and dependency-free. They can be installed on any Linux system which supports the `snapd` service (see the snapd documentation for more information). Security and robustness are their key features, alongside being easy to install, easy to maintain and easy to upgrade.

9.1.2 What else comes with it?

In addition to the upstream Kubernetes services, Canonical Kubernetes also includes:

- a DNS service for the node
- a CNI for the node/cluster
- a simple storage provider
- an ingress provider
- a load-balancer
- a gateway API controller
- a metrics server

9.1.3 Where can I install it?

The Canonical Kubernetes snap can be installed on a Linux OS, wherever it may be: run it in several local containers or VMs for example, or use it on public/private cloud instances. For deploying with Juju, a machine charm to deploy the snap is also available.

9.1.4 Can I use it to make a cluster?

Yes. Canonical Kubernetes is designed to be eminently scalable. You can start with a single node and add more as and when the need arises. Scale up or down at any time. Systems with more than three nodes will automatically become Highly Available.

9.1.5 Does it come with support?

Each and every user will be supported by the community. For a more detailed look at what that entails, please see our *Community page*. If you need a greater level of support, Canonical provides Ubuntu Pro, a comprehensive subscription for your open-source software stack. For more support options, visit the Ubuntu support page.

9.1.6 Next steps

- Try it now! Jump over to the *Getting started tutorial*

9.2 Channels

Canonical Kubernetes uses the concept of `channels` to make sure you always get the version of Kubernetes you are expecting, and that future upgrades can be handled with minimum, if any, disruption.

9.2.1 Choosing the right channel

When installing or updating Canonical Kubernetes you can (and should in most cases) specify a channel. The channel specified is made up of two components; the **track** and the **risk level**.

The track matches the minor version of upstream Kubernetes. For example, specifying the `1.30` track will match upstream releases of the same minor version ("1.30.0", "1.30.1", "1.30.x" etc.). Releases of Canonical Kubernetes closely follow the upstream releases and usually follow within 24 hours.

The 'risk level' component of the channel is one of the following:

- **`stable`**: Matches upstream stable releases
- **`candidate`**: Holds the release candidates of the snap
- **`beta`**: Tracks the beta releases - expect bugs
- **`edge`**: Experimental release including upstream alpha releases

Note that for each track, not all risk levels are guaranteed to be available. For example, there may be a new upstream version in development which only has an `edge` level. For a mature release, there may no longer be any `beta` or `candidate`. In these cases, if you specify a risk level which has no releases for that track the snap system will choose the closest available release with a lower risk level. Whatever risk level specified is the **maximum** risk level of the snap that will be installed - if you choose `candidate` you will never get `edge` for example.

For all snaps, you can find out what channels are available by running the `info` command, For example:

```
snap info k8s
```

More information can be found in the Snapcraft documentation

9.2.2 Updates and switching channels

Updates for upstream patch releases will happen automatically by default. For example, if you have selected the channel `1.30/stable`, your snap will refresh itself regularly keeping your cluster up-to-date with the latest patches. For deployments where this behaviour is undesirable you are given the option to postpone, schedule or even block automatic updates. The Snap refreshes documentation page outlines how to configure these options.

To change the channel of an already installed snap, the `refresh` command can be used:

```
sudo snap refresh k8s --channel=<new-channel>
```

> **Warning:** Changing the channel of an installed snap could result in loss of service. Please check any release notes or upgrade guides first!

9.2.3 Which channel is right for me?

Choosing the most appropriate channel for your needs depends on a number of factors. We can give some general guidance for the following cases:

- **I want to always be on the latest stable version matching a specific upstream K8s release (recommended).**

Specify the release, for example: `--channel=1.30/stable`.

- **I want to test-drive a pre-stable release**

Use `--channel=<next_release>/edge` for alpha releases.

Use `--channel=<next_release>/beta` for beta releases.

Use `--channel=<next_release>/candidate` for candidate releases.

- **I am waiting to test a bug fix on Canonical Kubernetes**

Use `--channel=<release>/edge`.

- **I am waiting for a bug fix from upstream Kubernetes**

Use `--channel=<release>/candidate`.

9.3 Security

This page provides an overview of various aspects of security to be considered when operating a cluster with **Canonical Kubernetes**. To consider security properly, this means not just aspects of Kubernetes itself, but also how and where it is installed and operated.

A lot of important aspects of security therefore lie outside the direct scope of **Canonical Kubernetes**, but links for further reading are provided.

9.3.1 Security of the snap/executable

As detailed in the snap documentation, an application installed from a snap is inherently more secure than a traditionally installed application. Snap-based applications are installed into a sandboxed, self contained environment which restricts its ability to interact with the rest of user space.

9.3.2 Security of the OCI images

Canonical Kubernetes relies on OCI standard images published as `rocks` to deliver the services which run and facilitate the operation of the Kubernetes cluster. The use of Rockcraft and `rocks` gives Canonical a way to maintain and patch images to remove vulnerabilities at their source, which is fundamental to our commitment to a sustainable Long Term Support(LTS) release of Kubernetes and overcoming the issues of stale images with known vulnerabilities. For more information on how these images are maintained and published, see the Rockcraft documentation.

9.3.3 Kubernetes Security

The Kubernetes cluster deployed by Canonical Kubernetes can be secured using any of the methods and options described by the upstream Kubernetes Security Documentation.

Canonical Kubernetes enables RBAC (Rules Based Access Control) by default.

9.3.4 Cloud security

If you are deploying **Canonical Kubernetes** on public or private cloud instances, anyone with credentials to the cloud where it is deployed may also have access to your cluster. Describing the security mechanisms of these clouds is out of the scope of this documentation, but you may find the following links useful.

- Amazon Web Services https://aws.amazon.com/security/
- Google Cloud Platform https://cloud.google.com/security/
- Metal As A Service(MAAS) https://maas.io/docs/snap/3.0/ui/hardening-your-maas-installation
- Microsoft Azure https://docs.microsoft.com/en-us/azure/security/azure-security
- VMWare VSphere https://www.vmware.com/security/hardening-guides.html

9.3.5 Security Compliance

As with previously released Kubernetes software from Canonical, we aim to satisfy the needs of various security compliance standards. This is a process that will take some time however. Please watch out for future announcements and check the *roadmap* for current areas of work.

For topics specifically relating to the snap version of Canonical Kubernetes, please see the *explanation topic*.

9.4 Other documentation types

If you are just getting started, the *Tutorials section* contains step-by-step tutorials to help guide you through exploring and using Canonical Kubernetes.

If you have a specific goal our *How-to guides* have more in-depth detail than tutorials and can be applied to a broader set of applications. They'll help you achieve an end-result but may require you to understand and adapt the steps to fit your specific requirements.

Finally, our *Reference section* is for when you need to check specific details or information such as the command reference or release notes.

CHAPTER

TEN

REFERENCE

Our Reference section is for when you need to check specific details or information such as the command reference or release notes.

10.1 Release notes

10.1.1 Rolling preview release

In advance of a GA release of Canonical Kubernetes, you can still install and try out the newest distribution of Kubernetes.

You need two commands to get a single node cluster, one for installation and another for cluster bootstrap. You can try it out now on your console by installing the k8s snap from the beta channel:

```
sudo snap install k8s --channel=1.30-classic/beta --classic
sudo k8s bootstrap
```

Currently Canonical Kubernetes is working towards general availability, but you can install it now to try:

- **Clustering** - need high availability or just an army of worker nodes? Canonical Kubernetes is emminently scaleable, see the *tutorial on adding more nodes*.
- **Networking** - Our built-in network component allows cluster administrators to automatically scale and secure network policies across the cluster. Find out more in our *how-to guides*.
- **Observability** - Canonical Kubernetes ships with COS Lite, so you never need to wonder what your cluster is actually doing. See the *observability documentation* for more details.

Follow along with the *tutorial* to get started!

10.2 Canonical Kubernetes charms

Canonical Kubernetes can be deployed via Juju with the following charms:

- **k8s**, which deploys a complete Kubernetes implementation
- **k8s-worker**, which deploys Kubernetes without the control plane for units intended as workers in a cluster (at least one `k8s` charm must be deployed and integrated with the worker for it to function)

10.2.1 Charmhub

Both of the above charms are published to Charmhub.

- The Charmhub page for the `k8s` charm
- The Charmhub page for the `k8s-worker` charm

For an explanation of the releases and channels, please see the documentation *explaining channels*.

10.2.2 Source

The source code for both charms is contained in a single repository:

https://github.com/canonical/k8s-operator

Please see the readme file there for further specifics of the charm implementation.

10.3 Proxy environment variables

Canonical Kubernetes uses the standard system-wide environment variables to control access through proxies.

On Ubuntu and other Linux operating systems, proxies are configured through system-wide environment variables defined in the `/etc/environment` file.

- **HTTPS_PROXY**
- **HTTP_PROXY**
- **NO_PROXY**
- **https_proxy**
- **http_proxy**
- **no_proxy**

10.3.1 No-proxy CIDRS

When configuring proxies, it is important to note that there are always some CIDRs which need to be excluded and added to the `no-proxy` lists. For Canonical Kubernetes these are:

- The range used by Kubernetes services (defaults to **10.152.183.0/24**)
- The range used by the Kubernetes pods (defaults to **10.1.0.0/16**)

And it is also important to exclude the local network to maintain access to any local traffic.

10.3.2 Configuring

For the `k8s` snap, proxy configuration is controlled by editing the `etc/environment` file mentioned above. There is an example in the *How to guide for configuring proxies for the k8s snap*.

For charms deployed by Juju, proxies are managed by configuring the model. See the *How to guide for configuring proxies for k8s charms* for an example of how to set these.

10.4 K8s charm arcitecture

Canonical `k8s` Charms encompass two primary components: the `k8s` charm and the `k8s-worker` charm.

Charms are instantiated on a machine as a Juju unit, and a collection of units constitutes an application. Both `k8s` and `k8s-worker` units are responsible for installing and managing its machine's `k8s` snap, however the charm type determines the node's role in the Kubernetes cluster. The `k8s` charm manages `control-plane` nodes, whereas the `k8s-worker` charm manages Kubernetes `worker` nodes. The administrator manages the cluster via the `juju` client, directing the `juju` controller to reach the model's eventually consistent state. For more detail on Juju's concepts, see the Juju docs.

The administrator may choose any supported cloud-types (Openstack, MAAS, AWS, GCP, Azure…) on which to manage the machines making up the Kubernetes cluster. Juju selects a single leader unit per application to act as a centralised figure with the model. The `k8s` leader oversees Kubernetes bootstrapping and enlistment of new nodes. Follower `k8s` units will join the cluster using secrets shared through relation data from the leader. The entire lifecycle of the deployment is orchestrated by the `k8s` charm, with tokens and cluster-related information being exchanged through Juju relation data.

Furthermore, the `k8s-worker` unit functions exclusively as a worker within the cluster, establishing a relation with the `k8s` leader unit and requesting tokens and cluster-related information through relation data. The `k8s` leader is responsible for issuing these tokens and revoking them if a unit administratively departs the cluster.

The `k8s` charm also supports the integration of other compatible charms, enabling integrations such as connectivity with an external `etcd` datastore and the sharing of observability data with the `Canonical Observability Stack (COS)`. This modular and integrated approach facilitates a robust and flexible Canonical Kubernetes deployment managed through Juju.

10.5 Welcome to the Canonical Kubernetes community

This rapidly growing community is a diverse bunch of people - developers, Kubernetes admins, inventors, researchers, students… and we all share the joy of a reliable, flexible, secure, timely version of upstream Kubernetes. The team recognise the important role each and every user plays in the success of the project as a whole and how valuable your contributions are.

10.5.1 Do you have questions?

Do you have questions about Canonical Kubernetes? Perhaps you want some ideas on how to best achieve a certain goal or maybe some aspect of your Kubernetes doesn't behave the way you expect. Perhaps you'd just like some advice from more experienced users. There are a number of ways to get in touch:

- Using the Kubernetes slack: find us in the #canonical-kubernetes channel
- In the public Matrix room
- On the Ubuntu Discourse

For more formal support, please see the support options available to you on the Ubuntu website.

Our commitment to you - we may not always be able to answer your questions, but we promise to respond within three working days.

10.5.2 Found a bug?

You can always track what is going on with development by watching our GitHub repository. This is also the best place to file a bug if you find one, or of course you are also welcome to contribute to the code.

Our commitment to you - we monitor the issues on GitHub regularly and we aim to triage all bug reports within three working days.

10.5.3 Contributing to the code?

Canonical Kubernetes is proudly open source, published under the GPLv3 license. We welcome and encourage contributions to the code. Please see the *Developer guide* for more information on contributing.

Our commitment to you - we closely follow activity on the source repository. We aim to respond to any PRs within three working days.

10.5.4 Contributing to docs?

Our documentation is extremely important to us and is actively maintained by the entire team. That doesn't mean that it can't be improved though. Every page in the documentation has an "Edit this page" link in the top right which takes you to GitHub to make small changes. For larger contributions, please see the *Contributing guide*.

Our commitment to you: Comments are usually read daily and we are really grateful for docs improvements.

10.5.5 Code of conduct

Building a fair, open and inclusive community is important to us. We think adopting a code of conduct is a sensible way to make sure that everybody participating understands what the expectations and obligations are. The team adopts the Ubuntu Code of Conduct 2.0, and we use these as the guidelines for participation.

10.6 Other documentation types

Alternatively, the *Tutorials section* contains step-by-step tutorials to help guide you through exploring and using Canonical Kubernetes.

If you have a specific goal our *How-to guides* have more in-depth detail than tutorials and can be applied to a broader set of applications. They'll help you achieve an end-result but may require you to understand and adapt the steps to fit your specific requirements.

Finally, for a better understanding of how Canonical Kubernetes works and related topics such as security, our *Explanation section* helps you to expand your knowledge and get the most out of Kubernetes.

Install K8s from a snap ›

Our tutorials, How To guides and other pages will explain how to install, configure and use the Canonical Kubernetes 'k8s' snap.

Deploy K8s using Juju ›

Our tutorials, How To guides and other pages will explain how to install, configure and use the Canonical Kubernetes 'k8s' charm.

CHAPTER

ELEVEN

PROJECT AND COMMUNITY

Canonical Kubernetes is a member of the Ubuntu family. It's an open source project which welcomes community involvement, contributions, suggestions, fixes and constructive feedback.

- Our Code of Conduct
- Our *community*
- How to *contribute*
- Our development *roadmap*