Take control of Kubernetes

How to use MicroK8s with Juju

Juju works well with MicroK8s to provide a full CNCF-certified Kubernetes system in under 60 seconds.

Contents:

Install and set up MicroK8s

> For installation instructions for other platforms, including Windows 10, macOS, ARM devices such as Raspberry Pi and LXD, see the MicroK8s documentation.

The easiest way to install MicroK8s is via `snap`:

```
sudo snap install microk8s --classic
```

Join the `microk8s` group

When strictly confined MicroK8s is released, the group you need to be a member of changes to `snap_microk8s` .

Add your account to the `microk8s` group. This grants the account elevated privileges to the cluster, meaning that `sudo` will not be required to interact with `microk8s`:

```
sudo usermod -a -G microk8s $USER
sudo chown -f -R $USER ~/.kube
```

This command requires you to log out and log back in before the changes are applied.

```
su - $USER
```

Alternatively, exit the shell by executing `exit` or by using `Ctrl` + `D` and restart your session.

Await installation

MicroK8s will take a few minutes to install all of its components.

```
microk8s status --wait-ready
```

Verify installation

Now let's inspect the cluster with the `microk8s.kubectl` command:

```
microk8s.kubectl get all --all-namespaces
```

You should see output similar to:

```
NAMESPACE   NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
default     service/kubernetes   ClusterIP   10.152.183.1   <none>        443/TCP   22s
```

It's possible that you encounter an error message relating to permissions. If this happens, execute `sudo usermod -a -G microk8s $USER` and restart.

```
Insufficient permissions to access MicroK8s.
You can either try again with sudo or add the user ubuntu to the 'microk8s' group:

    sudo usermod -a -G microk8s $USER

The new group will be available on the user's next login.
```

Enable DNS and storage add-ons

Now enable some MicroK8s addons for storage and DNS support:

```
microk8s.enable hostpath-storage dns
```

This will bring about changes to the cluster. Re-invoking the `microk8s.kubectl get all --all-namespaces` should eventually give you something like this:

```
NAMESPACE     NAME                                          READY   STATUS    RESTARTS   AGE
kube-system   pod/hostpath-provisioner-6d744c4f7c-t9hgh     1/1     Running   0          2m50s
kube-system   pod/kube-dns-6bfbdd666c-rxnp9                 3/3     Running   0          2m56s

NAMESPACE     NAME                  TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
default       service/kubernetes    ClusterIP   10.152.183.1     <none>        443/TCP
kube-system   service/kube-dns      ClusterIP   10.152.183.10    <none>        53/UDP,53/TCP

NAMESPACE     NAME                                     READY   UP-TO-DATE   AVAILABLE   AGE
kube-system   deployment.apps/hostpath-provisioner     1/1     1            1           2m50s
kube-system   deployment.apps/kube-dns                 1/1     1            1           2m56s

NAMESPACE     NAME                                                DESIRED   CURRENT   READY
kube-system   replicaset.apps/hostpath-provisioner-6d744c4f7c     1         1         1
kube-system   replicaset.apps/kube-dns-6bfbdd666c                 1         1         1
```

Install the Juju client

Just like MicroK8s, the Juju client is easiest to install via `snap`:

```
sudo snap install juju --classic
```

For other installation methods, see https://juju.is/docs/olm/installing-juju.

Create a controller

Juju recognises when MicroK8s is installed and automatically sets up a cloud called “microk8s”. There is no need to manually add the cluster to Juju (verify this with `juju clouds --local`). A controller can then be created just like a normal cloud. Here we’ve called it “micro”:

```
juju bootstrap microk8s micro
```

Important term - cloud: Within the Juju community, the term cloud has a specific meaning. A cloud is a target that Juju knows how to manage workloads for. Kubernetes clusters, including MicroK8s, are often referred to as “k8s clouds”.

Using a version of Juju earlier than v2.6.0: Earlier versions of Juju did not have built-in support for MicroK8s. You should upgrade your version of Juju to the current stable release. If that’s not possible, you can use `juju add-k8s` to register your MicroK8s cluster with Juju.

Verify the bootstrap process

Confirm the microk8s cloud is live by running `juju clouds`. It should be visible within the controller and client sections:

```
juju clouds
```

…which should result in output similar to:

```
Only clouds with registered credentials are shown.
There are more clouds, use --all to see them.

Clouds available on the controller:
Cloud     Regions  Default    Type
microk8s  1        localhost  k8s

Clouds available on the client:
Cloud           Regions  Default           Type       Credentials  Source    Description
[...]
microk8s        1        localhost         k8s        1            built-in  A Kubernetes C
[...]
```

Important concept - clouds: A cloud, as a Juju term, means a deployment target. It includes API endpoints and credential information. In Juju, you interact with the *client* (the `juju` command on your local machine). It connects to a *controller*. The controller is hosted on a *cloud* and controls *models*. A cloud can be registered on the controller, the client or both. Registering a cloud with the client allows the client to bootstrap a new controller onto it. Registering a cloud with a controller allows it to control models hosted on multiple providers.

Add a model

A model is a wrapper for a set of related applications and infrastructure. That model represents resources needed for the applications within it, including compute, storage, and network.

```
juju add-model testing
```

Difference from traditional clouds: There is no ‘default’ model provided on k8s clouds. Creating a model implies creating a Kubernetes namespace. Rather than pollute your cluster’s namespaces, Juju favours an explicit approach.

Verify that the model has been added

Use the `juju models` command to list models hosted on the cloud:

```
juju models
```

The output will list the available models, their type, status, and connection details for the current client.

```
Controller: micro

Model       Cloud/Region        Type        Status     Access  Last connection
controller  microk8s/localhost  kubernetes  available  admin   just now
testing*    microk8s            kubernetes  available  admin   never connected
```

Deploy a Kubernetes charm

We can now deploy a Kubernetes charm. For example, here we deploy a charm by requesting the use of the ‘mariadb-pv’ workload storage pool we just set up:

```
juju deploy cs:~juju/mariadb-k8s --storage database=10M
```

Verify deployment

The output to `juju status` should soon look like the following:

```
Model      Controller  Cloud/Region  Version    SLA          Timestamp
k8s-model  mk8s        microk8s      2.6-beta2  unsupported  14:47:20Z

App          Version  Status  Scale  Charm        Store       Rev  OS          Address
mariadb-k8s           active      1  mariadb-k8s  jujucharms    1  kubernetes  10.152.183.15

Unit            Workload  Agent  Address    Ports     Message
mariadb-k8s/0*  active    idle   10.1.1.15  3306/TCP
```

In contrast to standard Juju behaviour, there are no machines listed here. Let’s see what has happened within the cluster:

```
microk8s.kubectl get all --all-namespaces
```

New sample output:

```
NAMESPACE         NAME                                        READY   STATUS    RESTARTS   A
controller-mk8s   pod/controller-0                            2/2     Running   1          6
k8s-model         pod/mariadb-k8s-0                           1/1     Running   0          7
k8s-model         pod/mariadb-k8s-operator-0                  1/1     Running   0          9
kube-system       pod/hostpath-provisioner-6d744c4f7c-t9hgh   1/1     Running   0          9
kube-system       pod/kube-dns-6bfbdd666c-rxnp9               3/3     Running   0          9

NAMESPACE         NAME                            TYPE        CLUSTER-IP       EXTERNAL-IP   PO
controller-mk8s   service/controller-service      ClusterIP   10.152.183.168   <none>        17
default           service/kubernetes              ClusterIP   10.152.183.1     <none>        44
testing           service/mariadb-k8s             ClusterIP   10.152.183.153   <none>        33
kube-system       service/kube-dns                ClusterIP   10.152.183.10    <none>        53

NAMESPACE     NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
kube-system   deployment.apps/hostpath-provisioner    1/1     1            1           9m39s
kube-system   deployment.apps/kube-dns                1/1     1            1           9m45s

NAMESPACE     NAME                                               DESIRED   CURRENT   READY
kube-system   replicaset.apps/hostpath-provisioner-6d744c4f7c    1         1         1
kube-system   replicaset.apps/kube-dns-6bfbdd666c                1         1         1

NAMESPACE         NAME                                  READY   AGE
controller-mk8s   statefulset.apps/controller           1/1     6m6s
testing           statefulset.apps/mariadb-k8s          1/1     77s
testing           statefulset.apps/mariadb-k8s-operator   1/1     93s
```

You can easily identify the changes, as compared to the initial output, by scanning the left-hand side for our model name ‘k8s-model’, which runs in a Kubernetes namespace of the same name. The operator/controller pod runs in a namespace whose name is based on our controller name (‘mk8s’): “controller-mk8s”.

To get information on pod ‘mariadb-k8s-0’ you need to refer to the namespace (since it’s not the ‘default’ namespace) in this way:

```
microk8s.kubectl describe pods -n testing mariadb-k8s-0
```

The output is too large to sensibly include here. See the upstream documentation on different ways of viewing cluster information.

Remove configuration and software

To remove all traces of what we’ve done in this tutorial, use the following commands:

```
juju kill-controller -y -t 0 micro
microk8s.reset
sudo snap remove microk8s
sudo snap remove juju
```

That’s the end of this tutorial!

Next steps

- Learn more about using Kubernetes with Juju

Last updated 2 months ago.