Single node Kubernetes with MicroK8s

Install MicroK8s on Linux

MicroK8s installs a single node, CNCF-certified Kubernetes cluster in seconds. MicroK8s is a lightweight, zero-ops Kubernetes for Linux, Windows and macOS. A single command installs all upstream Kubernetes services and their dependencies. With support for x86 and ARM64, MicroK8s runs from local workstations to the edge and IoT appliances.

1. Install MicroK8s on Linux

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

2. Add your user to the microk8s admin group

MicroK8s creates a group to enable seamless usage of commands which require admin privilege. Use the following commands to join the group:

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

You will also need to re-enter the session for the group update to take place:

```
su - $USER
```

3. Check the status while Kubernetes starts

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

4. Turn on the services you want

```
microk8s enable dashboard dns ingress
```

Try `microk8s enable --help` for a list of available services and optional features. `microk8s disable ‹name›` turns off a service.

5. Start using Kubernetes

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

If you mainly use MicroK8s you can make our kubectl the default one on your command-line with `alias mkctl="microk8s kubectl"`. Since it is a standard upstream kubectl, you can also drive other Kubernetes clusters with it by pointing to the respective kubeconfig file via the "--kubeconfig" argument.

6. Access the Kubernetes dashboard

```
microk8s dashboard-proxy
```

7. Start and stop Kubernetes

Kubernetes is a collection of system services that talk to each other all the time. If you don't need them running in the background then you will save battery by stopping them. `microk8s start` and `microk8s stop` will do the work for you

Multi-node Clusters

Charmed Kubernetes installs CNCF-certified Kubernetes clusters across clouds

Charmed Kubernetes is a fully automated, model-driven approach to installing and managing Kubernetes from bare-metal to the cloud. Build your Kubernetes cloud from the ground up, integrate it with your favorite tools, and create multi-cloud topologies.

These instructions represent the complete set of commands you need to enter into your terminal to install Charmed Kubernetes on AWS.

1. Install LXD

LXD is a system container and VM hypervisor that allows you to create a local cloud. It can be installed via a snap package.

```
sudo snap install lxd --classic
```

2. Initialise LXD

LXD provides an interactive dialogue to configure your local cloud during the initialisation procedure:

```
lxd init
```

The init script itself may vary depending on the version of LXD. You can use most default options in the dialogue. The important configuration options for Charmed Kubernetes are:

- Networking: Do NOT enable ipv6 networking on the bridge interface
- Storage Pool: Use the ‘dir’ storage type

Running result:

```
Name of the new storage pool [default=default]:
Name of the storage backend to use (dir, lvm, powerflex, zfs, btrfs, ceph) [default=zfs]: dir
Would you like to connect to a MAAS server? (yes/no) [default=no]:
Would you like to create a new local network bridge? (yes/no) [default=yes]:
```

```
What should the new bridge be called? [default=lxdbr0]:
What IPv4 address should be used? (CIDR subnet notation, “auto” or “none”) [default=auto]:
What IPv6 address should be used? (CIDR subnet notation, “auto” or “none”) [default=auto]:
Would you like the LXD server to be available over the network? (yes/no) [default=no]:
Would you like stale cached images to be updated automatically? (yes/no) [default=yes]:
Would you like a YAML "lxd init" preseed to be printed? (yes/no) [default=no]:
```

3. Install Juju

Juju is a tool for deploying, configuring and operating complex software on public or private clouds. It can be installed with a snap:

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

4. Add juju controller

The Juju controller is used to manage the software deployed through Juju, from deployment to upgrades to day-two operations. One Juju controller can manage multiple projects or workspaces, which in Juju are known as ‘models’.

Juju comes preconfigured to work with LXD. A cloud created by using LXD containers on the local machine is known as localhost to Juju. To begin, you need to create a Juju controller for this cloud:

```
juju bootstrap localhost
```

Running result:

```
Creating Juju controller "localhost-localhost" on localhost/localhost
Looking for packaged Juju agent version 3.6.0 for amd64
Located Juju agent version 3.6.0-ubuntu-amd64 at https://streams.canonical.com/juju/tools/agent/3.6.0/juju-3.6.0-linux-amd64.tgz
To configure your system to better support LXD containers, please see:
https://documentation.ubuntu.com/lxd/en/latest/explanation/performance_tuning/
Launching controller instance(s) on localhost/localhost...
 - juju-77ea8f-0 (arch=amd64)
Installing Juju agent on bootstrap instance
Waiting for address
Attempting to connect to <ipv4-address>:22
Attempting to connect to [<ipv6-address>]:22
Connected to <ipv6-address>
Running machine configuration script...
Bootstrap agent now started
```

```
Contacting Juju controller at <ipv4-address> to verify accessibility...

Bootstrap complete, controller "localhost-localhost" is now available
Controller machines are in the "controller" model

Now you can run
        juju add-model <model-name>
to create a new model to deploy workloads.
```

5. Add a Kubernetes model

The model holds a specific deployment. It is a good idea to create a new one specifically for each deployment.

```
juju add-model k8s
```

Remember that you can have multiple models on each controller, so you can deploy multiple Kubernetes clusters or other applications.

Running result:

```
Added 'k8s' model on localhost/localhost with credential 'localhost' for user 'admin'
```

6. Deploy Kubernetes

Deploy the Kubernetes bundle to the model. This will add instances to the model and deploy the required applications. This can take up to 20 minutes depending on your machine.

```
juju deploy charmed-kubernetes
```

Running result:

```
Located bundle "charmed-kubernetes" in charm-hub, revision 1267
Located charm "calico" in charm-hub, channel latest/stable
Located charm "containerd" in charm-hub, channel latest/stable
Located charm "easyrsa" in charm-hub, channel latest/stable
Located charm "etcd" in charm-hub, channel latest/stable
Located charm "kubeapi-load-balancer" in charm-hub, channel latest/stable
Located charm "kubernetes-control-plane" in charm-hub, channel latest/stable
Located charm "kubernetes-worker" in charm-hub, channel latest/stable
Executing changes:
- upload charm calico from charm-hub for base ubuntu@22.04/stable from channel stable with
```

```
architecture=amd64
- deploy application calico from charm-hub on ubuntu@22.04/stable with stable
  added resource calico
  added resource calico-arm64
- set annotations for calico
- upload charm containerd from charm-hub for base ubuntu@22.04/stable from channel stable with
architecture=amd64
- deploy application containerd from charm-hub on ubuntu@22.04/stable with stable
  added resource containerd
- set annotations for containerd
- upload charm easyrsa from charm-hub for base ubuntu@22.04/stable from channel stable with
architecture=amd64
- deploy application easyrsa from charm-hub on ubuntu@22.04/stable with stable
  added resource easyrsa
- set annotations for easyrsa
- upload charm etcd from charm-hub for base ubuntu@22.04/stable from channel stable with
architecture=amd64
- deploy application etcd from charm-hub on ubuntu@22.04/stable with stable
  added resource core
  added resource etcd
  added resource snapshot
- set annotations for etcd
- upload charm kubeapi-load-balancer from charm-hub for base ubuntu@22.04/stable from channel stable with
architecture=amd64
- deploy application kubeapi-load-balancer from charm-hub on ubuntu@22.04/stable with stable
  added resource nginx-prometheus-exporter
- expose all endpoints of kubeapi-load-balancer and allow access from CIDRs 0.0.0.0/0 and ::/0
- set annotations for kubeapi-load-balancer
- upload charm kubernetes-control-plane from charm-hub for base ubuntu@22.04/stable from channel stable
with architecture=amd64
- deploy application kubernetes-control-plane from charm-hub on ubuntu@22.04/stable with stable
  added resource cni-plugins
- set annotations for kubernetes-control-plane
- upload charm kubernetes-worker from charm-hub for base ubuntu@22.04/stable from channel stable with
architecture=amd64
- deploy application kubernetes-worker from charm-hub on ubuntu@22.04/stable with stable
  added resource cni-plugins
- expose all endpoints of kubernetes-worker and allow access from CIDRs 0.0.0.0/0 and ::/0
- set annotations for kubernetes-worker
- add relation kubernetes-control-plane:loadbalancer-external - kubeapi-load-balancer:lb-consumers
- add relation kubernetes-control-plane:loadbalancer-internal - kubeapi-load-balancer:lb-consumers
- add relation kubernetes-control-plane:kube-control - kubernetes-worker:kube-control
- add relation kubernetes-control-plane:certificates - easyrsa:client
- add relation etcd:certificates - easyrsa:client
- add relation kubernetes-control-plane:etcd - etcd:db
- add relation kubernetes-worker:certificates - easyrsa:client
- add relation kubeapi-load-balancer:certificates - easyrsa:client
- add relation calico:etcd - etcd:db
- add relation calico:cni - kubernetes-control-plane:cni
- add relation calico:cni - kubernetes-worker:cni
- add relation containerd:containerd - kubernetes-worker:container-runtime
- add relation containerd:containerd - kubernetes-control-plane:container-runtime
- add unit easyrsa/0 to new machine 0
- add unit etcd/0 to new machine 1
- add unit etcd/1 to new machine 2
- add unit etcd/2 to new machine 3
- add unit kubeapi-load-balancer/0 to new machine 4
- add unit kubernetes-control-plane/0 to new machine 5
- add unit kubernetes-control-plane/1 to new machine 6
- add unit kubernetes-worker/0 to new machine 7
- add unit kubernetes-worker/1 to new machine 8
- add unit kubernetes-worker/2 to new machine 9
Deploy of bundle completed.
```

7. Monitor the deployment

Juju is now busy creating instances, installing software and connecting the different parts of the cluster together, which can take several minutes. You can monitor what’s going on by running:

```
watch -c juju status --color
```

To view the last twenty log messages for the “k8s” model:

```
juju debug-log -m k8s -n 20
```

8. Start using your cluster!

Congratulations! You have a Kubernetes cluster up and running - now let’s use it!

Multi-node, highly available Kubernetes with MicroK8s

To create a cluster out of two or more already-running MicroK8s instances, use the microk8s add-node command. As of MicroK8s 1.19, clustering of three or more nodes will automatically enable high availability.

The MicroK8s instance on which the command is run will host the Kubernetes control plane:

```
microk8s add-node
```

The add-node command prints a microk8s join command which should be executed on the MicroK8s instance(s) that you wish to join to the cluster (NOT THE NODE YOU RAN add-node FROM). For example:

```
microk8s join ip-172-31-20-243:25000/DDOkUupkmaBezNnMheTBqFYHLWINGDbf
```

Joining a node to the cluster should only take a few seconds. Afterwards you should be able to see the node has joined:

```
microk8s kubectl get no
```