100 XP

Exercise - Explore the functionality of a Kubernetes cluster

15 minutes

Several options are available when you're running Kubernetes locally. You can install Kubernetes on physical machines or VMs, or use a cloud-based solution such as Azure Kubernetes Service (AKS).

Your goal in this exercise is to explore a Kubernetes installation with a single-node cluster. You're going to configure a *MicroK8s* environment that's easy to set up and tear down. Then, you'll deploy an NGINX website and scale it out to multiple instances. Finally, you'll go through the steps to delete the running pods and clean up the cluster.

> **Note**
>
> This exercise is optional and includes steps that show how to delete and uninstall the software and resources you'll use in the exercise.

Keep in mind that there are other options, such as MiniKube and Kubernetes support in Docker, to do the same.

What is MicroK8s?

MicroK8s is an option for deploying a single-node Kubernetes cluster as a single package to target workstations and Internet of Things (IoT) devices. Canonical, the creator of Ubuntu Linux, originally developed and still maintains MicroK8s.

You can install MicroK8s on Linux, Windows, and macOS. However, installation instructions are slightly different for each operating system. Choose the option that best fits your environment.

Install MicroK8s on macOS

To run MicroK8s on macOS, use Multipass. Multipass is a lightweight VM manager for Linux, Windows, and macOS.

1. You have two options to install Multipass on macOS. Either download and install the latest release of Multipass for macOS from GitHub , or to install Multipass with the `brew cask install multipass` command, use Homebrew.

   Bash

   ```
   brew install --cask multipass
   ```

2. In a command console, run the multipass launch command to configure and run the microk8s-vm image. This step might take a few minutes to complete, depending on the speed of your internet connection and desktop.

   Console

   ```
   multipass launch --name microk8s-vm --memory 4G --disk 40G
   ```

3. After you receive the launch confirmation for microk8s-vm, run the `multipass shell microk8s-vm` command to enter the VM instance.

   Console

   ```
   multipass shell microk8s-vm
   ```

   At this point, you can access the Ubuntu VM that will host your cluster. You still have to install MicroK8s. Follow these steps.

4. Install the MicroK8s snap app. This step might take a few minutes to complete, depending on the speed of your internet connection and desktop.

   Bash

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

A successful installation shows the following message:

Output

```
2020-03-16T12:50:59+02:00 INFO Waiting for restart...
microk8s v1.17.3 from Canonical✓ installed
```

You're now ready to install add-ons on the cluster.

Prepare the cluster

To view the status of the installed add-ons on your cluster, run the status command in MicroK8s. These add-ons provide several services, some of which you covered previously. One example is DNS functionality.

1. To check the status of the installation, run the `microk8s.status --wait-ready` command.

Bash

```
sudo microk8s.status --wait-ready
```

Notice that you can enable several add-ons on your cluster. Don't worry about the add-ons that you don't recognize. You'll enable only three of these add-ons in your cluster.

Output

```
microk8s is running
addons:
cilium: disabled
dashboard: disabled
dns: disabled
fluentd: disabled
gpu: disabled
helm3: disabled
helm: disabled
ingress: disabled
istio: disabled
jaeger: disabled
juju: disabled
knative: disabled
kubeflow: disabled
linkerd: disabled
metallb: disabled
metrics-server: disabled
prometheus: disabled
rbac: disabled
registry: disabled
storage: disabled
```

2. Next, you'll enable the DNS, Dashboard, and Registry add-ons. Here's the purpose of each add-on:

Add-ons	Purpose
DNS	Deploys the `coreDNS` service.
Dashboard	Deploys the `kubernetes-dashboard` service and several other services that support its functionality. It's a general-purpose, web-based UI for Kubernetes clusters.
Registry	Deploys a private registry and several services that support its functionality. To store private containers, use this registry.

To install the add-ons, run the following command.

Bash

```
sudo microk8s.enable dns dashboard registry
```

You're now ready to access your cluster by running `kubectl`.

Explore the Kubernetes cluster

MicroK8s provides a version of `kubectl` that you can use to interact with your new Kubernetes cluster. This copy of `kubectl` allows you to have a parallel installation of another system-wide `kubectl` instance without affecting its functionality.

1. Run the `snap alias` command to alias `microk8s.kubectl` to `kubectl`. This step simplifies usage.

   Bash

   ```
   sudo snap alias microk8s.kubectl kubectl
   ```

   The following output appears when the command finishes successfully:

   Output

   ```
   Added:
     - microk8s.kubectl as kubectl
   ```

Display cluster node information

Recall from earlier that a Kubernetes cluster exists out of control planes and worker nodes. Let's explore the new cluster to see what's installed.

1. Check the nodes that are running in your cluster.

   You know that MicroK8s is a single-node cluster installation, so you expect to see only one node. Keep in mind, though, that this node is both the control plane and a worker node in the cluster. Confirm this configuration by running the `kubectl get nodes` command. To retrieve information about all the resources in your cluster, run the `kubectl get` command:

   Bash

   ```
   sudo kubectl get nodes
   ```

   The result is similar to the following example, which shows you that there's only one node in the cluster with the name microk8s-vm. Notice that the node is in a ready state. The ready state indicates that the control plane might schedule workloads on this node.

   Output

   ```
   NAME          STATUS   ROLES    AGE   VERSION
   microk8s-vm   Ready    <none>   35m   v1.17.3
   ```

You can get more information for the specific resource that's requested. For example, let's assume that you need to find the IP address of the node. To fetch extra information from the API server, run the `-o wide` parameter:

Bash

```
sudo kubectl get nodes -o wide
```

The result is similar to the following example. Notice that you now can see the internal IP address of the node, the OS running on the node, the kernel version, and the container runtime.

Output

```
NAME          STATUS   ROLES    AGE   VERSION   INTERNAL-IP      EXTERNAL-IP   OS-IMAGE
KERNEL-VERSION      CONTAINER-RUNTIME
microk8s-vm   Ready    <none>   36m   v1.17.3   192.168.56.132   <none>        Ubuntu 18.04.4 LTS
```

```
4.15.0-88-generic   containerd://1.2.5
```

2. The next step is to explore the services running on your cluster. As with nodes, to find information about the services running on the cluster, run the `kubectl get` command.

Bash

```
sudo kubectl get services -o wide
```

The result is similar to the following example, but notice that only one service is listed. You installed add-ons on the cluster earlier, and you'd expect to see these services as well.

Output

```
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE   SELECTOR
kubernetes   ClusterIP   10.152.183.1   <none>        443/TCP   37m   <none>
```

The reason for the single service listing is that Kubernetes uses a concept called *namespaces* to logically divide a cluster into multiple virtual clusters.

To fetch all services in all namespaces, pass the `--all-namespaces` parameter:

Bash

```
sudo kubectl get services -o wide --all-namespaces
```

The result is similar to the following example. Notice that you have three namespaces in your cluster. They're the default, `container-registry`, and `kube-system` namespaces. Here, you can see the `registry`, `kube-dns`, and `kubernetes-dashboard` instances that you installed. You'll also see the supporting services that were installed alongside some of the add-ons.

Output

```
NAMESPACE           NAME                        TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE   SELECTOR
container-registry   registry                   NodePort    10.152.183.36    <none>
5000:32000/TCP          28m   app=registry
default              kubernetes                 ClusterIP   10.152.183.1     <none>        443/TCP
37m   <none>
kube-system          dashboard-metrics-scraper   ClusterIP   10.152.183.130   <none>
8000/TCP                28m   k8s-app=dashboard-metrics-scraper
kube-system          heapster                   ClusterIP   10.152.183.115   <none>        80/TCP
28m   k8s-app=heapster
kube-system          kube-dns                   ClusterIP   10.152.183.10    <none>
53/UDP,53/TCP,9153/TCP   28m   k8s-app=kube-dns
kube-system          kubernetes-dashboard       ClusterIP   10.152.183.132   <none>        443/TCP
28m   k8s-app=kubernetes-dashboard
kube-system          monitoring-grafana         ClusterIP   10.152.183.88    <none>        80/TCP
28m   k8s-app=influxGrafana
kube-system          monitoring-influxdb        ClusterIP   10.152.183.232   <none>
8083/TCP,8086/TCP       28m   k8s-app=influxGrafana
```

Now that you can see the services running on the cluster, you can schedule a workload on the worker node.

Install a web server on a cluster

You want to schedule a web server on the cluster to serve a website to your customers. You can choose from several options. For this example, you'll use NGINX.

Recall from earlier that you can use pod manifest files to describe your pods, replica sets, and deployments to define workloads. Because you haven't covered these files in detail, you'll use `kubectl` to directly pass the information to the API server.

Even though the use of `kubectl` is handy, using manifest files is a best practice. Manifest files allow you to roll forward or roll back deployments with ease in your cluster. These files also help document the configuration of a cluster.

1. To create your NGINX deployment, run the `kubectl create deployment` command. Specify the name of the deployment and the container image to create a single instance of the pod.

Bash

```
sudo kubectl create deployment nginx ––image=nginx
```

The result is similar to the following example:

Output

```
deployment.apps/nginx created
```

2. To fetch the information about your deployment, run `kubectl get deployments`:

Bash

```
sudo kubectl get deployments
```

The result is similar to the following example. Notice that the name of the deployment matches the name you gave it, and that one deployment with this name is in a ready state and available.

Output

```
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   1/1     1            1           18s
```

3. The deployment created a pod. To fetch info about your cluster's pods, run the `kubectl get pods` command:

Bash

```
sudo kubectl get pods
```

The result is similar to the following example. Notice that the name of the pod is a generated value prefixed with the name of the deployment, and the pod has a status of *Running*.

Output

```
NAME                     READY   STATUS    RESTARTS   AGE
nginx-86c57db685-dj6lz   1/1     Running   0          33s
```

Test the website installation

Test the NGINX installation by connecting to the web server through the pod's IP address.

1. To find the address of the pod, pass the `-o wide` parameter:

Bash

```
sudo kubectl get pods -o wide
```

The result is similar to the following example. Notice that the command returns both the IP address of the node, and the node name on which the workload is scheduled.

Output

```
NAME                     READY   STATUS    RESTARTS   AGE     IP           NODE          NOMINATED
NODE   READINESS GATES
nginx-86c57db685-dj6lz   1/1     Running   0          4m17s   10.1.83.10   microk8s-vm   <none>
<none>
```

2. To access the website, run `wget`:

Bash

```
wget 10.1.83.10
```

The result is similar to the following example:

Output

```
--2020-03-16 13:34:17--  http://10.1.83.10/
Connecting to 10.1.83.10:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 612 [text/html]
Saving to: 'index.html'

index.html
100%[=======================================================================================
>]     612  --.-KB/s    in 0s

2020-03-16 13:34:17 (150 MB/s) - 'index.html' saved [612/612]
```

Scale a web server deployment on a cluster

Assume that you suddenly see an increase in users who access your website, and the website starts failing because of the load. You can deploy more instances of the site in your cluster and split the load across the instances.

To scale the number of replicas in your deployment, run the `kubectl scale` command. You specify the number of replicas you need and the name of the deployment.

1. To scale the total of NGINX pods to three, run the `kubectl scale` command:

   Bash

   ```
   sudo kubectl scale --replicas=3 deployments/nginx
   ```

   The result is similar to the following example:

   Output

   ```
   deployment.apps/nginx scaled
   ```

   The scale command allows you to scale the instance count up or down.

2. To check the number of running pods, run the `kubectl get` command, and again pass the `-o wide` parameter:

   Bash

   ```
   sudo kubectl get pods -o wide
   ```

   The result is similar to the following example. Notice that you now see three running pods, each with a unique IP address.

   Output

   ```
   NAME                     READY   STATUS    RESTARTS   AGE     IP           NODE          NOMINATED
   NODE   READINESS GATES
   nginx-86c57db685-dj6lz   1/1     Running   0          7m57s   10.1.83.10   microk8s-vm   <none>
   <none>
   nginx-86c57db685-lzrwp   1/1     Running   0          9s      10.1.83.12   microk8s-vm   <none>
   <none>
   nginx-86c57db685-m7vdd   1/1     Running   0          9s      10.1.83.11   microk8s-vm   <none>
   <none>
   ubuntu@microk8s-vm:~$
   ```

You'd need to apply several more configurations to the cluster to effectively expose your website as a public-facing website. Examples include installing a load balancer and mapping node IP addresses. This type of configuration forms part of advanced aspects that you'll explore in the future.

Uninstall MicroK8s

To recover space on your development machine, you can remove everything you've deployed so far, even the VM. Keep in mind that this procedure is optional.

1. To remove the add-ons from the cluster, run the `microk8s.disable` command, and specify the add-ons to remove:

```
Bash

sudo microk8s.disable dashboard dns registry
```

2. To remove MicroK8s from the VM, run the `snap remove` command:

```
Bash

sudo snap remove microk8s
```

If you want to remove the Multipass VM manager from your machine, there are a few extra steps to take on Windows and macOS.

1. To exit the VM, run the `exit` command:

```
Bash

exit
```

2. To stop the VM, run the `multipass stop` command and specify the VM's name:

```
Bash

multipass stop microk8s-vm
```

3. To delete and purge the VM instance, run `multipass delete`, then run `multipass purge`:

```
Console

multipass delete microk8s-vm
multipass purge
```

Next unit: When to use Kubernetes

Continue >