



# Exercise - Explore the functionality of a Kubernetes cluster

15 minutes

#### Choose your platform

Linux	Mac	Windows
-------	-----	---------

Several options are available when you're running Kubernetes locally. Recall that 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 and explore a single-node Kubernetes 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 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.

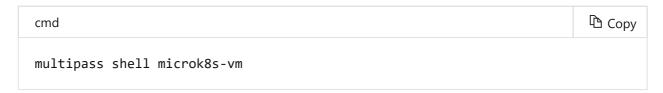
#### Install MicroK8s on Windows

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

- 1. Download and install the latest release of Multipass for Windows from GitHub 2.
- 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.



3. After you receive the launch confirmation for microk8s-vm, you can access the VM instance by using the multipass shell microk8s-vm command.

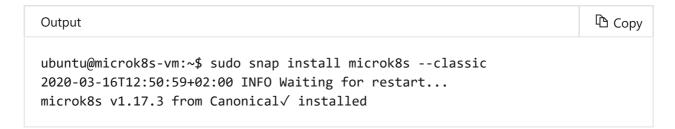


At this point, you can access the Ubuntu VM that will host your cluster and install MicroK8s.

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



A successful installation shows the following message.



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

# Prepare the cluster

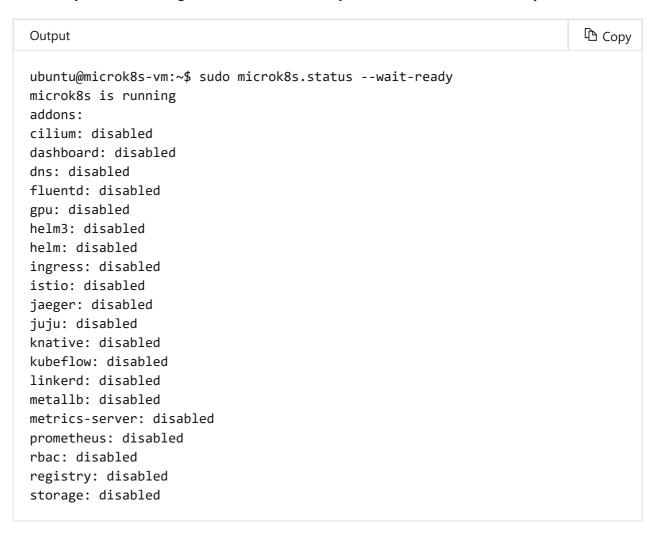
You can use the status command in MicroK8s to view the status of the installed add-ons on your cluster. These add-ons provide several services, some of which you covered previously.

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.



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

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. You can use this registry to store private containers.

Install the add-ons by running the following command.



You're now ready to access your cluster by using kubect1.

# **Explore the Kubernetes cluster**

MicroK8s provides a version of kubect1 that you can use to interact with your new Kubernetes cluster. This copy of kubect1 allows you to have a parallel installation of another system-wide kubect1 instance without affecting its functionality.

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



You'll see the following output when the command finishes successfully.



## Display cluster node information

Recall from earlier that a Kubernetes cluster exists out of master 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. You can use the kubectl get command to retrieve information about all the resources in your cluster.



```
sudo kubectl get nodes
```

The result will be 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.

```
Bash

ubuntu@microk8s-vm:~$ sudo kubectl get nodes

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. You use the -o wide parameter to fetch extra information from the API server.

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

The result will be 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.

```
Bash
                                                                  Copy
ubuntu@microk8s-vm:~$ sudo kubectl get nodes -o wide
NAME
            STATUS ROLES AGE VERSION
                                                            EXTERNAL-IP
                                            INTERNAL-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, you can use the kubectl get command to find information about the services running on the cluster.

```
Bash

sudo kubectl get services -o wide
```

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

Bash						Copy
ubuntu@micro	ok8s-vm:∼\$ su TYPE	udo kubectl get CLUSTER-IP	services -o w		AGE	
SELECTOR kubernetes	ClusterIP	10.152.183.1	<none></none>	443/TCP	37m	<none></none>

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

Use the --all-namespaces parameter to fetch all services in all namespaces.

```
Bash

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

The result will be 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.

Bash							D Col
ubuntu@microl	(8s-vm:	\$ sudo kubectl get	t servi	ices	-o widea	ll-namespac	es
NAMESPACE		NAME			TYPE	CLUSTER-I	Р
EXTERNAL-IP	PORT(S	5)	AGE	SEL	ECTOR		
container-reg	gistry	registry			NodePort	10.152.18	3.36
<none></none>	5000:3	32000/TCP	28m	арр	=registry		
default		kubernetes			ClusterIP	10.152.18	3.1
<none></none>	443/TC	CP CP	37m	<no< td=""><td>ne&gt;</td><td></td><td></td></no<>	ne>		
kube-system		dashboard-metrics	s-scrap	er	ClusterIP	10.152.18	3.130
<none></none>	8000/1	СР	28m	k8s	-app=dashboa	ard-metrics	-
scraper							
kube-system		heapster			ClusterIP	10.152.18	3.115
<none></none>	80/TCF		28m	k8s	-app=heapste	er	
kube-system		kube-dns			ClusterIP	10.152.18	3.10
<none></none>	53/UDF	,53/TCP,9153/TCP	28m	k8s	-app=kube-dr	าร	
kube-system		kubernetes-dashbo	pard		ClusterIP	10.152.18	3.132
<none></none>	443/TC	CP CP	28m	k8s	-app=kuberne	etes-dashbo	ard
kube-system		monitoring-grafa	าล		ClusterIP	10.152.18	3.88
<none></none>	80/TCF	)	28m	k8s	-app=influx(	Grafana	
kube-system		monitoring-influx	kdb		ClusterIP	10.152.18	3.232
<none></none>	8083/1	CP,8086/TCP		k8s	-app=influx(	Grafana	

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.

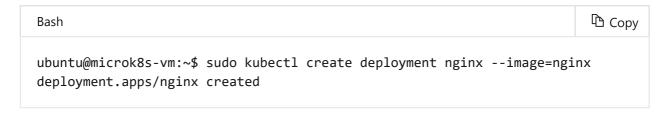
Recall from earlier that you can use pod manifest files to describe your pods, replica sets, and deployments to define workloads. Because you haven't covered these files in detail, you'll use kubectl to directly pass the information to the API server.

Even though the use of kubect1 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. Use the kubect1 create deployment command to create your NGINX deployment. Specify the name of the deployment and the container image to create a single instance of the pod.



The result will be similar to the following example.



2. Use kubectl get deployments to fetch the information about your deployment.



The result will be 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.

```
Bash

ubuntu@microk8s-vm:~$ sudo kubectl get deployments

NAME READY UP-TO-DATE AVAILABLE AGE

nginx 1/1 1 1 18s
```

3. The deployment created a pod. Use the kubectl get pods command to fetch info about your cluster's pods.



The result will be 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.

```
Bash

ubuntu@microk8s-vm:~$ sudo kubectl get pods

NAME READY STATUS RESTARTS AGE

nginx-86c57db685-dj6lz 1/1 Running 0 33s

ubuntu@microk8s-vm:~$
```

#### Test the website installation

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

1. Use the -o wide parameter to find the address of the pod.

```
Bash

ubuntu@microk8s-vm:∼$ sudo kubectl get pods -o wide
```

The result will be 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.



2. Use wget to access the website.

```
Bash

wget 10.1.83.10

□ Copy
```

The result will be similar to the following example.

```
Bash

ubuntu@microk8s-vm:~$ wget 10.1.83.10
--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

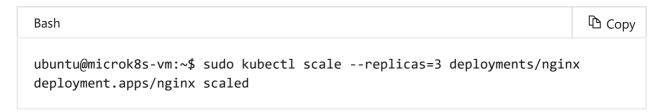
Assume that you suddenly see an increase in users who access your website, and the website starts failing because of the load. You can deploy more instances of the site in your cluster and split the load across the instances.

You can use the kubectl scale command to scale the number of replicas in your deployment. You specify the number of replicas you need and the name of the deployment.

1. Run the kubectl scale command to scale the total of NGINX pods to three.



The result will be similar to the following example.



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

2. Check the number of running pods by using the kubectl get command, and again pass the -o wide parameter.

```
Bash

ubuntu@microk8s-vm:~$ sudo kubectl get pods -o wide
```

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

Bash					🕒 Сору			
ubuntu@microk8s-vm:~\$ sudo kubectl get pods -o wide								
NAME	READY	STATUS	RESTARTS	AGE	IP			
NODE NOMINATE	D NODE	READINESS	GATES					
nginx-86c57db685-dj6lz	1/1	Running	0	7m57s	10.1.83.10			
microk8s-vm <none></none>		<none></none>						
nginx-86c57db685-lzrwp	1/1	Running	0	9s	10.1.83.12			
microk8s-vm <none></none>		<none></none>						
nginx-86c57db685-m7vdd	1/1	Running	0	9s	10.1.83.11			
microk8s-vm <none></none>		<none></none>						
ubuntu@microk8s-vm:~\$								

You would need to apply several additional 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.

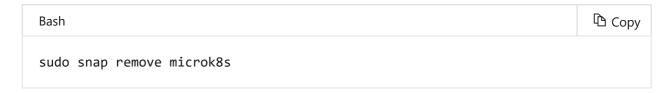
# **Uninstall MicroK8s**

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

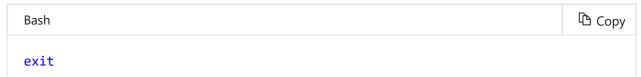
1. Remove the add-ons from the cluster by running the microk8s.disable command and specifying the add-ons to remove.



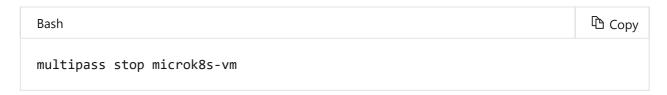
2. Remove MicroK8s from the VM by running the snap remove command.



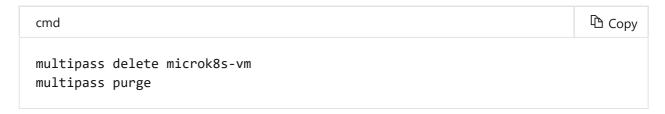
3. Exit the VM by running the exit command.



4. Stop the VM by running the multipass stop command and specifying the VM's name.



5. Delete and purge the VM instance by running multipass delete and then multipass purge.



## Next unit: When to use Kubernetes

Continue >