The portability and reproducibility of a containerized process mean we have an opportunity to move and scale our containerized applications across clouds and datacenters. Containers effectively guarantee that those applications run the same way anywhere, allowing us to quickly and easily take advantage of all these environments. Furthermore, as we scale our applications up, we’ll want some tooling to help automate the maintenance of those applications, able to replace failed containers automatically, and manage the rollout of updates and reconfigurations of those containers during their lifecycle.

Tools to manage, scale, and maintain containerized applications are called orchestrators, and the most common examples of these are Kubernetes and Docker Swarm. Development environment deployments of both of these orchestrators are provided by Docker Desktop, which we’ll use throughout this guide to create our first orchestrated, containerized application.

The advanced modules teach you how to:

The advanced modules teach you how to:

1. [Set up and use a Kubernetes environment on your development machine](https://docs.docker.com/get-started/kube-deploy/)
2. [Set up and use a Swarm environment on your development machine](https://docs.docker.com/get-started/swarm-deploy/)

Enable Kubernetes

Docker Desktop will set up Kubernetes for you quickly and easily. Follow the setup and validation instructions appropriate for your operating system:

### **Windows**

1. After installing Docker Desktop, you should see a Docker icon in your system tray. Right-click on it, and navigate **Settings** > **Kubernetes**.
2. Check the checkbox labeled **Enable Kubernetes**, and click **Apply & Restart**. Docker Desktop will automatically set up Kubernetes for you. You’ll know that Kubernetes has been successfully enabled when you see a green light beside ‘Kubernetes running’ in the **Settings** menu.
3. In order to confirm that Kubernetes is up and running, create a text file called pod.yaml with the following content:
4. apiVersion: v1
5. kind: Pod
6. metadata:
7. name: demo
8. spec:
9. containers:
10. - name: testpod
11. image: alpine:3.5
12. command: ["ping", "8.8.8.8"]

This describes a pod with a single container, isolating a simple ping to 8.8.8.8.

1. In PowerShell, navigate to where you created pod.yaml and create your pod:
2. kubectl apply -f pod.yaml
3. Check that your pod is up and running:
4. kubectl get pods

You should see something like:

NAME READY STATUS RESTARTS AGE

demo 1/1 Running 0 4s

1. Check that you get the logs you’d expect for a ping process:
2. kubectl logs demo

You should see the output of a healthy ping process:

PING 8.8.8.8 (8.8.8.8): 56 data bytes

64 bytes from 8.8.8.8: seq=0 ttl=37 time=21.393 ms

64 bytes from 8.8.8.8: seq=1 ttl=37 time=15.320 ms

64 bytes from 8.8.8.8: seq=2 ttl=37 time=11.111 ms

...

1. Finally, tear down your test pod:

kubectl delete -f pod.yaml

## Enable Docker Swarm

Docker Desktop runs primarily on Docker Engine, which has everything you need to run a Swarm built in. Follow the setup and validation instructions appropriate for your operating system:

* [Mac](https://docs.docker.com/get-started/orchestration/#swarmosx)
* [Windows](https://docs.docker.com/get-started/orchestration/#swarmwin)

### **Windows**

1. Open a powershell, and initialize Docker Swarm mode:
2. docker swarm init

If all goes well, you should see a message similar to the following:

Swarm initialized: current node (tjjggogqpnpj2phbfbz8jd5oq) is now a manager.

To add a worker to this swarm, run the following command:

docker swarm join --token SWMTKN-1-3e0hh0jd5t4yjg209f4g5qpowbsczfahv2dea9a1ay2l8787cf-2h4ly330d0j917ocvzw30j5x9 192.168.65.3:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

1. Run a simple Docker service that uses an alpine-based filesystem, and isolates a ping to 8.8.8.8:
2. docker service create --name demo alpine:3.5 ping 8.8.8.8
3. Check that your service created one running container:
4. docker service ps demo

You should see something like:

ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS

463j2s3y4b5o demo.1 alpine:3.5 docker-desktop Running Running 8 seconds ago

1. Check that you get the logs you’d expect for a ping process:
2. docker service logs demo

You should see the output of a healthy ping process:

demo.1.463j2s3y4b5o@docker-desktop | PING 8.8.8.8 (8.8.8.8): 56 data bytes

demo.1.463j2s3y4b5o@docker-desktop | 64 bytes from 8.8.8.8: seq=0 ttl=37 time=13.005 ms

demo.1.463j2s3y4b5o@docker-desktop | 64 bytes from 8.8.8.8: seq=1 ttl=37 time=13.847 ms

demo.1.463j2s3y4b5o@docker-desktop | 64 bytes from 8.8.8.8: seq=2 ttl=37 time=41.296 ms

...

1. Finally, tear down your test service:
2. docker service rm demo

## Conclusion

At this point, you’ve confirmed that you can run simple containerized workloads in Kubernetes and Swarm. The next step will be to write the Kubernetes yaml that describes how to run and manage these containers on Kubernetes.