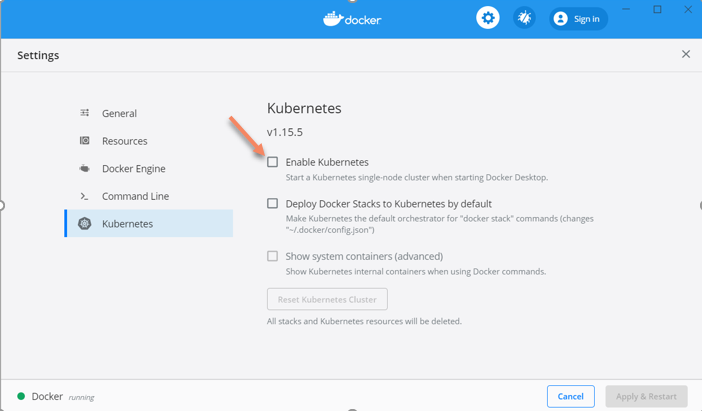
**Deploying Applications to Kubernetes**

1. Docker Desktop
2. Minikube

### **Kubernetes Installation**

A local machine Kubernetes solution can help developers to configure and run a Kubernetes cluster in their local development environments and test their application during all development phases, without investing significant effort to configure and manage a Kubernetes cluster.

Docker Desktop for Windows and Mac includes a standalone Kubernetes server that runs on our Windows host, so that we can test deploying our Docker workloads on Kubernetes.



* To enable Kubernetes support and install a standalone instance of Kubernetes running as a Docker container, select **Enable Kubernetes**.
* This instantiates images required to run the Kubernetes server as containers, and installs the **kubectl.exe** command in the path. If we have kubectl already installed and pointing to some other environment, such as minikube, be sure to change context so that kubectl is pointing to docker-desktop:
* When Kubernetes is enabled and running, an additional status bar item displays at the bottom right of the Docker Desktop Settings dialog. The status of Kubernetes shows in the Docker menu and the context points to **docker-desktop** (Kubernetes cluster)
* To delete all stacks and Kubernetes resources, select **Reset Kubernetes Cluster**.
* To disable Kubernetes support at any time, clear the **Enable Kubernetes** check box. The Kubernetes containers are stopped and removed, and the /usr/local/bin/kubectl command is removed.

**Note**: By default, Kubernetes containers are hidden from commands like **docker service ls**, because managing them manually is not supported. To make them visible, check "Show system containers (advanced)" checkbox under Kubernetes menu.

**Testing the installation**

We can run a quick and easy test, to make sure that Kubernetes is actually running on the machine. Open command prompt / terminal window and run the command:

kubectl version

**To get detailed information about the cluster:**

kubectl cluster-info

Kubernetes should report that both Kubernetes master and KubeDNS are running on localhost:6443

Kubectl performs all its operations against the current context:

> kubectl config get-contexts

Note: Following files contains all Clusters and Contexts information

Windows: C:\Users\<user-name>\.kube\**config**

Linux/Mac: /home/<username>/.kube/**config**

To set the current context:

kubectl config **use-context**  docker-desktop

**View the cluster and context configuration**

kubectl config view

Note that the above command shows the content of the file C:\Users\<user-name>\.kube\**config OR** /home/training/.kube/config

# **Setting up the Kubernetes tooling on Windows 10 WSL**

<https://itnext.io/setting-up-the-kubernetes-tooling-on-windows-10-wsl-d852ddc6699c>

Online Emulator: <https://labs.play-with-k8s.com/>

**Installing Minikube on Ubuntu**

**Update System and install packages**

sudo apt-get update -y

sudo apt-get upgrade -y

sudo apt-get install curl

sudo apt-get install apt-transport-https

**Install VirtualBox Hypervisor**

sudo apt install virtualbox virtualbox-ext-pack

**Install Minikube**

wget https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

sudo cp minikube-linux-amd64 /usr/local/bin/minikube

sudo chmod 755 /usr/local/bin/minikube

minikube version

**Install Kubectl**

curl -LO https://storage.googleapis.com/kubernetes-release/release/`curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt`/bin/linux/amd64/kubectl

chmod +x ./kubectl

sudo mv ./kubectl /usr/local/bin/kubectl

kubectl version -o json

**Start Minikube**

minikube start

kubectl config view

kubectl cluster-info

kubectl get nodes

kubectl get pod

**Other Minikube comands**

minikube status

minikube stop

minikube delete

minikube addons list

minikube dashboard

**Reference:** [**https://phoenixnap.com/kb/install-minikube-on-ubuntu**](https://phoenixnap.com/kb/install-minikube-on-ubuntu)

**Installing Minikube on Windows**

Docker Desktop for Windows/Mac uses Type-1 hypervisor such as Hyper-V, which are better compared to Type-2 hypervisors, such as VirtualBox. Minikube supports both hypervisors. Unfortunately, there are limitations in which technology we are using, since we cannot have Type-1 or Type-2 hypervisors running at the same time on our machine:

Hyper-V can run on three versions of Windows 10: Windows 10 Enterprise, Windows 10 Professional, and Windows 10 Education.

**Step1) Install a Hypervisor**

If we do not already have a hypervisor installed, install one of these:

* Hyper-V
* VirtualBox

**Step2:** Install Chocolatey package manager for Windows.

For installation of Chocolatey, use the following command from PowerShell in administrative mode:

PS:> Set-ExecutionPolicy Bypass -Scope Process -Force; iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))

**Step3:** We can **install kubectl** according to the instructions available at

<https://kubernetes.io/docs/tasks/tools/install-kubectl/#install-kubectl-on-windows>

**Option 1)** Install Kubectl.exe using Chocolatey command.

***choco install kubernetes-cli***

## ****Step4: Minikube Installation****

Install Minikube using Chocolatey: The easiest way to install Minikube on Windows is using Chocolatey (run as an administrator):

**C:\> choco install minikube**

After Minikube has finished installing, close the current CLI session and restart. Minikube should have been added to our path automatically.

**Step4: Start Minikube and create a cluster:**

**C:\>** minikube start

**C:\>** minikube status

**C:\>** minikube stop

**Command to redirect docker cli to minikube host (On Windows / Mac with Docker Desktop installed)**

echo $(minikube docker-env)

eval $(minikube docker-env)

docker ps

Creating a Cluster in AWS EC2 Instances

# Using kubeadm

## Creating a kubernetes cluster in AWS with kubeadm

We're going to create a single master node and single worker node cluster.

### Step 1: Set up the AWS infrastructure

First, make sure that the nodes are in a subnet and security group that allows the necessary traffic. Make sure the following ports are open:

\* 80

\* 8080

\* 6443

Launch two instances in the subnet. Choose a t3.medium for the master and a t3.small for the worker. I'm going to use Ubuntu.

> \*NOTE\*: To save some typing, you can install the software and then create an AMI to launch the rest of the nodes. I'll point out where to do this below.

### Step 2: Install the software on the instances

SSH into the master node

Set up the repo

```

$ sudo apt-get update

$ sudo apt-get install -y apt-transport-https

$ sudo su -

$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add

$ cat <<EOF > /etc/apt/sources.list.d/kubernetes.list

deb http://apt.kubernetes.io/ kubernetes-xenial main

EOF

$ apt-get update

```

Install Docker

```

$ apt-get install -y docker.io

```

Install kubeadm, kubectl, and kubernetes-cni

```

$ apt-get install -y kubelet kubeadm kubectl kubernetes-cni

```

> \*NOTE\*: At this point all the necessary software is installed. If you chose to go the AMI option above, then go create an AMI from this instance at this point. Then use that AMI to launch your worker node.

### Step 3: Initialize the master node

```

kubeadm init

```

The command will install Kubernetes onto the master node and will output some instructions, including configuring the kubectl command. Follow the first part where you set up kubectl. Leave the rest.

```

$ exit

$ mkdir -p $HOME/.kube

$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

$ sudo chown $(id -u):$(id -g) $HOME/.kube/config

```

Now kubectl will work and point to your cluster. Let's take a look at the nodes.

```

kubectl get nodes

```

Notice that the master node shows as `NotReady`. This is because we haven't installed a pod network as mentioned in the last step.

Let's install a network to allow the pods to communicate with each other.

```

$ sudo sysctl net.bridge.bridge-nf-call-iptables=1

net.bridge.bridge-nf-call-iptables = 1

$ export kubever=$(kubectl version | base64 | tr -d '\n')

$ kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$kubever"

serviceaccount/weave-net created

clusterrole.rbac.authorization.k8s.io/weave-net created

clusterrolebinding.rbac.authorization.k8s.io/weave-net created

role.rbac.authorization.k8s.io/weave-net created

rolebinding.rbac.authorization.k8s.io/weave-net created

daemonset.extensions/weave-net created

```

With the network installed, it may take a minute, but then you should see the master node marked as `Ready`

```

$ kubectl get nodes

NAME STATUS ROLES AGE VERSION

ip-172-31-25-218 Ready master 8m12s v1.14.3

```

We should also now see the dns pods running. Before they were not.

```

$ kubectl get pod --all-namespaces

NAMESPACE NAME READY STATUS RESTARTS AGE

kube-system coredns-fb8b8dccf-vwz2g 1/1 Running 0 9m11s

kube-system coredns-fb8b8dccf-zpsw4 1/1 Running 0 9m11s

kube-system etcd-ip-172-31-25-218 1/1 Running 0 8m4s

kube-system kube-apiserver-ip-172-31-25-218 1/1 Running 0 8m1s

kube-system kube-controller-manager-ip-172-31-25-218 1/1 Running 0 8m2s

kube-system kube-proxy-24pfq 1/1 Running 0 9m12s

kube-system kube-scheduler-ip-172-31-25-218 1/1 Running 0 8m24s

kube-system weave-net-772tf 2/2 Running 0 2m11s

```

### Step 4: Create and join the worker node

> \*NOTE\*: If you didn't create your worker instance from the AMI, then repeat step 2 above to install the necessary software.

SSH into the worker node

Join the worker node to the cluster. You can find the `kubeadm` command in the `kubeadm init` output from step 3 above. If you didn't capture that, this command will give it to you. Run this on the master node.

```

$ kubeadm token create --print-join-command

kubeadm join 172.31.25.218:6443 --token 463fwk.q91ecy4aogzov6rl --discovery-token-ca-cert-hash sha256:b379b12ced73ded8f9c2e3628f6025274e8052a758deccd64fc804411369e86d

```

Now run this command on your worker node

```

$ kubeadm join 172.31.25.218:6443 --token 463fwk.q91ecy4aogzov6rl --discovery-token-ca-cert-hash sha256:b379b12ced73ded8f9c2e3628f6025274e8052a758deccd64fc804411369e86d

[preflight] Running pre-flight checks

[WARNING Service-Docker]: docker service is not enabled, please run 'systemctl enable docker.service'

[WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup driver. The recommended driver is "systemd". Please follow the guide at https://kubernetes.io/docs/setup/cri/

[preflight] Reading configuration from the cluster...

[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'

[kubelet-start] Downloading configuration for the kubelet from the "kubelet-config-1.14" ConfigMap in the kube-system namespace

[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"

[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"

[kubelet-start] Activating the kubelet service

[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:

\* Certificate signing request was sent to apiserver and a response was received.

\* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

```

Now check that the nodes are ready

```

$ kubectl get nodes

The connection to the server localhost:8080 was refused - did you specify the right host or port?

```

Ah! If we want to run kubectl from the worker node, we need to configure it there as well. Or we can switch over and run the command on the master node as we did before.

```

$ kubectl get nodes

NAME STATUS ROLES AGE VERSION

ip-172-31-25-218 Ready master 23m v1.14.3

ip-172-31-5-74 Ready <none> 2m16s v1.14.3

```

Looking good. We have our master and our worker working. Let's also take a look at all the pods in our cluster to make sure everything is running clean.

```

$ kubectl get pods --all-namespaces

NAMESPACE NAME READY STATUS RESTARTS AGE

kube-system coredns-fb8b8dccf-vwz2g 1/1 Running 0 23m

kube-system coredns-fb8b8dccf-zpsw4 1/1 Running 0 23m

kube-system etcd-ip-172-31-25-218 1/1 Running 0 22m

kube-system kube-apiserver-ip-172-31-25-218 1/1 Running 0 22m

kube-system kube-controller-manager-ip-172-31-25-218 1/1 Running 0 22m

kube-system kube-proxy-24pfq 1/1 Running 0 23m

kube-system kube-proxy-wbwnc 1/1 Running 0 2m47s

kube-system kube-scheduler-ip-172-31-25-218 1/1 Running 0 22m

kube-system weave-net-45b2q 2/2 Running 1 2m47s

kube-system weave-net-772tf 2/2 Running 0 16m

```

### Step 5: Put a workload on the cluster

To verify that our cluster is working, let's install some pods. Let's put a nginx deployment on it and see how it shakes out.

```

$ cat <<EOF > nginx.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

labels:

app: nginx

spec:

replicas: 4

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx

EOF

$ kubectl create -f nginx.yaml

deployment.apps/nginx-deployment created

$ kubectl get all

NAME READY STATUS RESTARTS AGE

pod/nginx-deployment-7c995b74d9-b259m 1/1 Running 0 15s

pod/nginx-deployment-7c995b74d9-k77rr 1/1 Running 0 15s

pod/nginx-deployment-7c995b74d9-qck5c 1/1 Running 0 15s

pod/nginx-deployment-7c995b74d9-t29q7 1/1 Running 0 15s

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

service/kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 160m

NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/nginx-deployment 4/4 4 4 15s

NAME DESIRED CURRENT READY AGE

replicaset.apps/nginx-deployment-7c995b74d9 4 4 4 15s

```

As you can see, we now have 4 nginx pods running successfully.

# Final thoughts

At this point we have a working cluster. It is not production ready, and lacks some refinement. But it works. Pods, deployments, services, etc. will technically run. What it lacks now is the interaction with the cloud provider. In my case I used AWS. If I create a service with type `LoadBalancer`, it will not automatically provision an Elastic Load Balancer for me and hook it up with my cluster. This requires quite a bit more configuration.