Create a Kubernetes cluster using minikube

Install Minikube on Mac CLI

To install minikube on x86–64 mac OS, run the following two commands:

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-darwin-amd64
sudo install minikube-darwin-amd64 /usr/local/bin/minikube
```

Use the following command to start a cluster:

```
minikube start
```

Use the following command to confirm minikube is running:

```
minikube status
```

```
Brandis-MBP:W17complex brandi$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

Create a 3 Node Cluster

We want to create a 3 node cluster that will consist of one master node (for the rest of the tutorial we will call this node “control-plane”) and two worker nodes. We can create a 3 node cluster by using the following command:

```
minikube start --nodes # -p <cluster_name>
```

```
Use "minikube options" for a list of global command-line options (applies to all commands).
Brandis-MBP:W17complex brandi$ minikube start --nodes 3 -p k8cluster
😄  [k8cluster] minikube v1.29.0 on Darwin 12.6.3
✨  Automatically selected the docker driver
📌  Using Docker Desktop driver with root privileges
👍  Starting control plane node k8cluster in cluster k8cluster
🚜  Pulling base image ...
🔥  Creating docker container (CPUs=2, Memory=2200MB) ...
🐳  Preparing Kubernetes v1.26.1 on Docker 20.10.23 ...
    ▪ Generating certificates and keys ...
    ▪ Booting up control plane ...
    ▪ Configuring RBAC rules ...
🔗  Configuring CNI (Container Networking Interface) ...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🔎  Verifying Kubernetes components...
🌟  Enabled addons: storage-provisioner, default-storageclass

👍  Starting worker node k8cluster-m02 in cluster k8cluster
🚜  Pulling base image ...
🔥  Creating docker container (CPUs=2, Memory=2200MB) ...
🌐  Found network options:
    ▪ NO_PROXY=192.168.94.2
❗  This container is having trouble accessing https://registry.k8s.io
💡  To pull new external images, you may need to configure a proxy: https://minikube.sigs.k8s.io/docs/reference/networking/proxy/
🐳  Preparing Kubernetes v1.26.1 on Docker 20.10.23 ...
    ▪ env NO_PROXY=192.168.94.2
🔎  Verifying Kubernetes components...

👍  Starting worker node k8cluster-m03 in cluster k8cluster
🚜  Pulling base image ...
🔥  Creating docker container (CPUs=2, Memory=2200MB) ...
🌐  Found network options:
    ▪ NO_PROXY=192.168.94.2,192.168.94.3
🐳  Preparing Kubernetes v1.26.1 on Docker 20.10.23 ...
    ▪ env NO_PROXY=192.168.94.2
    ▪ env NO_PROXY=192.168.94.2,192.168.94.3
🔎  Verifying Kubernetes components...
🏄  Done! kubectl is now configured to use "k8cluster" cluster and "default" namespace by default
```

The cluster may take several minutes to create, but minikube will output updates as things are happening. Once it says “Done!” use the following command to see the three nodes you just created.

```
kubectl get nodes
```

```
Brandis-MBP:W17complex brandi$ kubectl get nodes
NAME           STATUS   ROLES           AGE   VERSION
k8cluster      Ready    control-plane   14m   v1.26.1
k8cluster-m02  Ready    <none>          12m   v1.26.1
k8cluster-m03  Ready    <none>          10m   v1.26.1
```

3 node cluster

Label Nodes

When we deploy our Redis and Apache pods, we do not want them to deploy to our control-plane, so we need to label our second and third nodes as “worker”. Use the following command to apply a worker label to the k8cluster-m02 and k8cluster-m03 nodes. You will have to run the command twice, once for each of the above nodes.

```
kubectl label node <node_name> node-role.kubernetes.io/worker=worker
```

```
Brandis-MBP:W17complex brandi$ kubectl label node k8cluster-m02 node-role.kubernetes.io/worker=worker
node/k8cluster-m02 labeled
Brandis-MBP:W17complex brandi$ kubectl label node k8cluster-m03 node-role.kubernetes.io/worker=worker
node/k8cluster-m03 labeled
```

Use the following command to view the newly labeled nodes.

```
kubectl get nodes
```

```
Brandis-MBP:W17complex brandi$ kubectl get nodes
NAME           STATUS   ROLES           AGE   VERSION
k8cluster      Ready    control-plane   18m   v1.26.1
k8cluster-m02  Ready    worker          15m   v1.26.1
k8cluster-m03  Ready    worker          13m   v1.26.1
```

The YAML files that we will deploy will search for these worker nodes based on a key:value label pair. Use the following command to apply a key:value label to the worker nodes:

```
kubectl label nodes <node_name> role=worker
```

```
Brandis-MBP:W17complex brandi$ kubectl label nodes k8cluster-m02 role=worker
node/k8cluster-m02 labeled
```

Run the command again for the second worker node (k8cluster-m03).

```
[Brandis-MBP:W17complex brandi$ kubectl label nodes k8cluster-m03 role=worker
 node/k8cluster-m03 labeled
```

Redis Deployment YAML File

In my last tutorial *Create a Kubernetes Cluster Using Docker Desktop*, I went into detail on how to create a Kubernetes deployment YAML file, If you need to create a YAML file for the Redis and Apache deployments, please visit that tutorial now and come back when your files are created. Alternatively, you can just copy the code below for the Redis deployment with four replicas and save it an a .yml file.

```
apiVersion: apps/v1
kind: Deployment
metadata:
    name: redis-deploy
    labels:
        app: redis
spec:
    replicas: 4
    selector:
        matchLabels:
            app: redis
    template:
        metadata:
            labels:
                app: redis
        spec:
            containers:
            - name: redis
              image: redis:7.0.9
              ports:
              - containerPort: 6379
            nodeSelector:
                role: worker
```

Notice in the file we've added **nodeSelector** on the container being created. With nodeSelector, we can choose which node our containers will be deployed on. Earlier we assigned our second and third nodes to be "worker" nodes and indicated that with the key:value pair role:worker. When we deploy this YAML file, the engine will look for any nodes that have the role:worker label and will deploy replicas on those nodes only. By using these labels, we are avoiding deploying replicas on the control-panel.

Apache Deployment YAML File

Here is the file contents for the Apache deployment with ten replicas.

```
apiVersion: apps/v1
kind: Deployment
metadata:
    name: httpd-deploy
    labels:
        app: httpd
spec:
    replicas: 10
    selector:
        matchLabels:
            app: httpd
    template:
        metadata:
            labels:
                app: httpd
        spec:
            containers:
            - name: httpd
              image: httpd:2.4.56
              ports:
              - containerPort: 80
            nodeSelector:
                role: worker
```

Note that these two files can actually be combined into one file by using `---` to separate each manifest. To see what asingle file would look like, visit my GitHub here.

Deploy Redis and Apache Pods

Using the CLI, navigate to the directory that contains your Redis and Apache YAML files. Use the following command to deploy the pods:

```
kubectl apply -f <yaml_file.yml>
```

You will need to run this command twice. Once for the Redis YAML file and once for the Apache YAML file.

```
Brandis-MBP:W17complex brandi$ kubectl apply -f k8_httpd.yml
deployment.apps/httpd-deploy created
Brandis-MBP:W17complex brandi$ kubectl apply -f k8s_redis.yml
deployment.apps/redis-deploy created
```

Your pods containing your replicas should have successfully deployed. To confirm this, use the following command:

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

```
Brandis-MBP:W17complex brandi$ kubectl get pods -o wide
NAME                            READY   STATUS    RESTARTS   AGE     IP           NODE           NOMINATED NODE   READINESS GATES
httpd-deploy-6765c96878-4wbb7   1/1     Running   0          2m45s   10.244.2.4   k8cluster-m03   <none>           <none>
httpd-deploy-6765c96878-5fjnb   1/1     Running   0          2m46s   10.244.2.6   k8cluster-m03   <none>           <none>
httpd-deploy-6765c96878-5gjmg   1/1     Running   0          2m45s   10.244.1.8   k8cluster-m02   <none>           <none>
httpd-deploy-6765c96878-6d4m6   1/1     Running   0          2m45s   10.244.2.8   k8cluster-m03   <none>           <none>
httpd-deploy-6765c96878-7cdkl   1/1     Running   0          2m45s   10.244.1.7   k8cluster-m02   <none>           <none>
httpd-deploy-6765c96878-896qx   1/1     Running   0          2m45s   10.244.2.7   k8cluster-m03   <none>           <none>
httpd-deploy-6765c96878-cfrs6   1/1     Running   0          2m45s   10.244.1.6   k8cluster-m02   <none>           <none>
httpd-deploy-6765c96878-lfhsr   1/1     Running   0          2m45s   10.244.1.5   k8cluster-m02   <none>           <none>
httpd-deploy-6765c96878-q4n7k   1/1     Running   0          2m45s   10.244.2.5   k8cluster-m03   <none>           <none>
httpd-deploy-6765c96878-shjpd   1/1     Running   0          2m45s   10.244.1.4   k8cluster-m02   <none>           <none>
redis-deploy-6ff456db6f-98jgg   1/1     Running   0          3m48s   10.244.2.3   k8cluster-m03   <none>           <none>
redis-deploy-6ff456db6f-d4dmj   1/1     Running   0          3m48s   10.244.1.3   k8cluster-m02   <none>           <none>
redis-deploy-6ff456db6f-fptv2   1/1     Running   0          3m48s   10.244.1.2   k8cluster-m02   <none>           <none>
redis-deploy-6ff456db6f-gbcdb   1/1     Running   0          3m48s   10.244.2.2   k8cluster-m03   <none>           <none>
```

You can see that all fourteen replicas are running and they are only running on the worker nodes (k8cluster-m02 and k8cluster-m03).

To get more information about what we just created, use the following command:

```
kubectl get all
```

```
[Brandis-MBP:W17complex brandi$ kubectl get all
NAME                                 READY   STATUS    RESTARTS   AGE
pod/httpd-deploy-6765c96878-4wbb7    1/1     Running   0          4m54s
pod/httpd-deploy-6765c96878-5fjnb    1/1     Running   0          4m55s
pod/httpd-deploy-6765c96878-5gjmg    1/1     Running   0          4m54s
pod/httpd-deploy-6765c96878-6d4m6    1/1     Running   0          4m54s
pod/httpd-deploy-6765c96878-7cdkl    1/1     Running   0          4m54s
pod/httpd-deploy-6765c96878-896qx    1/1     Running   0          4m54s
pod/httpd-deploy-6765c96878-cfrs6    1/1     Running   0          4m54s
pod/httpd-deploy-6765c96878-lfhsr    1/1     Running   0          4m54s
pod/httpd-deploy-6765c96878-q4n7k    1/1     Running   0          4m54s
pod/httpd-deploy-6765c96878-shjpd    1/1     Running   0          4m54s
pod/redis-deploy-6ff456db6f-98jgg    1/1     Running   0          5m57s
pod/redis-deploy-6ff456db6f-d4dmj    1/1     Running   0          5m57s
pod/redis-deploy-6ff456db6f-fptv2    1/1     Running   0          5m57s
pod/redis-deploy-6ff456db6f-gbcdb    1/1     Running   0          5m57s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   35m

NAME                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/httpd-deploy    10/10   10           10          4m55s
deployment.apps/redis-deploy    4/4     4            4           5m57s

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/httpd-deploy-6765c96878    10        10        10      4m55s
replicaset.apps/redis-deploy-6ff456db6f    4         4         4       5m57s
[Brandis-MBP:W17complex brandi$ kubectl get nodes
NAME           STATUS   ROLES           AGE   VERSION
k8cluster      Ready    control-plane   36m   v1.26.1
k8cluster-m02  Ready    worker          34m   v1.26.1
k8cluster-m03  Ready    worker          32m   v1.26.1
```

With this command you can view all your replicas, your cluster IP that was automatically created, your two deployments, your replicaSets that were created from your YAML files, and your three nodes. You have now successfully created a Kubernetes cluster with one control-plane node and two worker nodes that are running fourteen pods.

Clean Up

Clean up is optional but is a good practice to save space on your local computer. To clean up what you just created, we need to delete the deployments and the nodes. You can delete both deployments in a single command:

```
kubectl delete deployments <httpd_deployment_name> <redis_deployment_name>
```

```
[Brandis-MBP:W17complex brandi$ kubectl delete deployments httpd-deploy redis-deploy
 deployment.apps "httpd-deploy" deleted
 deployment.apps "redis-deploy" deleted
```

To delete your cluster nodes, use the following command:

```
kubectl delete nodes <node_name> <node_name> <node_name>
```

```
[Brandis-MBP:W17complex brandi$ kubectl delete nodes k8cluster k8cluster-m02 k8cluster-m03
 node "k8cluster" deleted
 node "k8cluster-m02" deleted
 node "k8cluster-m03" deleted
```

To stop minikube, run the following command:

```
minikube stop
```

To delete minikube, run the following command:

```
minikube delete
```