Initializing the Kubernetes Master Node

49m 56s left

Open Cloud Environment
✓ 100%
Setup completed
Average setup time: 3m 29s

Credentials

Account ID
030925358 Copy

Username
student Copy

Password
Ca1_6eFlz Copy

Region
US West 2 Copy

PEM Download
PPK Download

Bridge Connection
✓ 100%
Completed

Lab Steps

1 Logging In t Amazon W Services Co

Introduction

You will use `kubeadm` to initialize the control-plane node in this lab step. The initialization process will create a certificate authority for secure cluster communication and authentication, and start all the node components (`kubelet`), control-plane components (API server, controller manager, scheduler, etcd), and common add-ons (`kube-proxy`, DNS). You will see how easy the initialization process is with `kubeadm`.

The initialization uses sensible default values that adhere to best practices. However, many command options are available to configure the process, including if you want to provide your own certificate authority or if you want to use an external etcd key-value store. One option that you will use is required by the pod network plugin that you will install after the control-plane is initialized. `kubeadm` does not install a default network plugin for you. You will use Calico as the pod network plugin. Calico supports Kubernetes network policies. For network policies to function properly, you must use the `--pod-network-cidr` option to specify a range of IP addresses for the pod network when initializing the control-plane node with `kubeadm`.

There are many network plugins besides Calico. Calico is used primarily because it is used in clusters in Kubernetes certification exams and it supports network policies. Calico is used internally by AWS, Azure, and GCP for their managed Kubernetes offerings, so you can be certain it is production-ready. However, if all of your environments live in AWS, you may consider the Amazon VPC network plugin.

Instructions

1. Initialize the control-plane node using the init command:

Copy code

```
sudo kubeadm init --pod-network-cidr=192.168.0.0/1
```

Support

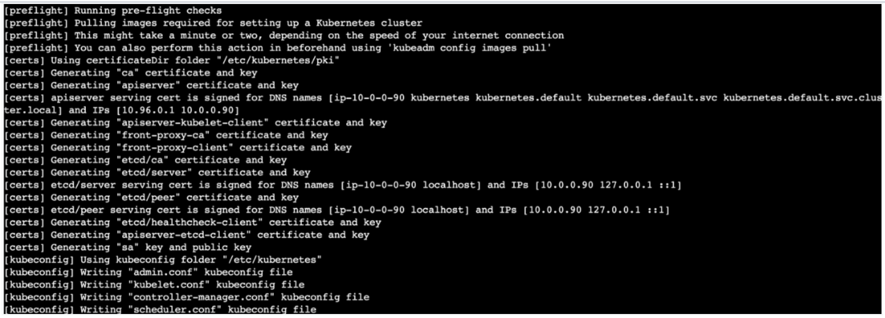
The pod network CIDR block (192.168.0.0/16) is the default used by

network CIDR used

onfiguration of

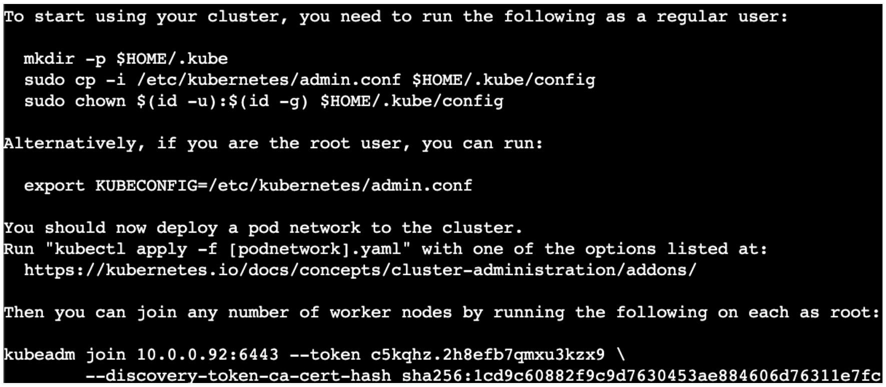
Skip to content Press option + Q to open this menu

Need help? Contact our support team



```
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [ip-10-0-0-90 kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.clus
ter.local] and IPs [10.96.0.1 10.0.0.90]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [ip-10-0-0-90 localhost] and IPs [10.0.0.90 127.0.0.1 ::1]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [ip-10-0-0-90 localhost] and IPs [10.0.0.90 127.0.0.1 ::1]
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "sa" key and public key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "kubelet.conf" kubeconfig file
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
```

Read through the output to understand what is happening. At the end of the output, useful commands for configuring `kubectl` and joining worker nodes to the cluster are given:



```
To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 10.0.0.92:6443 --token c5kqhz.2h8efb7qmxu3kzx9 \
        --discovery-token-ca-cert-hash sha256:1cd9c60882f9c9d7630453ae884606d76311e7fc
```

2. Copy the `kubeadm join` command at the end of the output and store it somewhere you can access later.

It is simply convenient to reuse the given command, although you can regenerate it and create new tokens using the `kubeadm token` command. The join tokens expire after 24 hours by default.

3. Initialize your user's default `kubectl` configuration using the admin kubeconfig file generated by `kubeadm`:

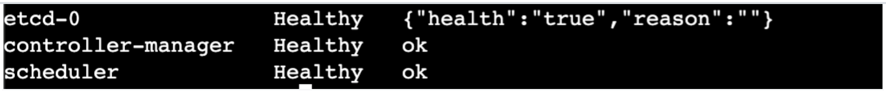
Copy code

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

4. Confirm you can use `kubectl` to get the cluster component statuses:

Skip to content Press option + Q to open this menu

Copy code



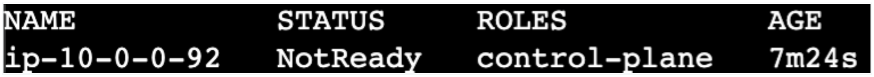
```
etcd-0               Healthy   {"health":"true","reason":""}
controller-manager   Healthy   ok
scheduler            Healthy   ok
```

The output confirms that the **scheduler**, **controller-manager**, and **etcd** are all **Healthy**. The Kubernetes API server is also operational, or `kubectl` would have returned an error attempting to connect to the API server.
Enter `kubeadm token --help` if you would like to know more about `kubeadm` tokens.

5. Get the nodes in the cluster:

Copy code

```
kubectl get nodes
```



```
NAME           STATUS     ROLES           AGE
ip-10-0-0-92   NotReady   control-plane   7m24s
```

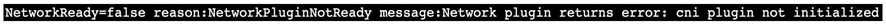
The control-plane node is reporting a **STATUS** of **NotReady**. Notice `kubeadm` gives the node a **NAME** based on its IP address. The `--node-name` option can be used to override the default behavior.

6. Describe the node to probe deeper into its NotReady status:

Copy code

```
kubectl describe nodes
```

In the **Conditions** section of the output, observe the **Ready** condition is False, and read the **Message**:



```
NetworkReady=false reason:NetworkPluginNotReady message:Network plugin returns error: cni plugin not initialized
```

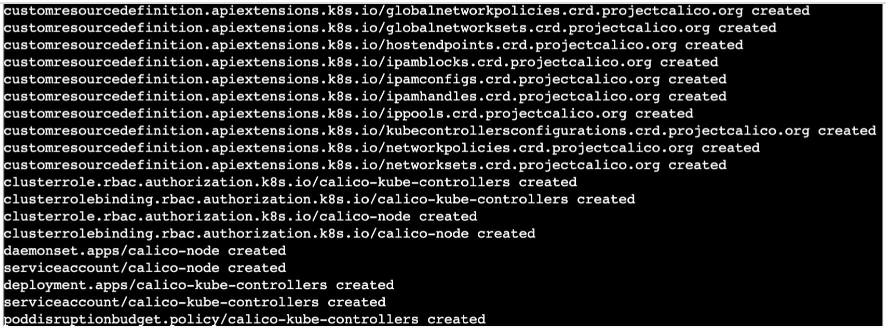
The kubelet is not ready because the network plugin is not ready. The **cni config uninitialized** refers to the container network interface (CNI) and is a related problem. Network plugins implement the CNI interface. You will resolve the issue by initializing the Calico network plugin.

7. Enter the following commands to create the Calico network plugin for pod networking:

Copy code

```
us-west-2.amazona
```

Skip to content Press option + Q to open this menu



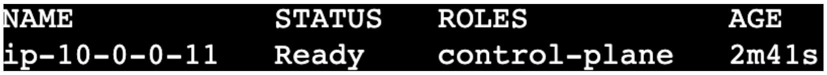
```
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/kubecontrollersconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
daemonset.apps/calico-node created
serviceaccount/calico-node created
deployment.apps/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
poddisruptionbudget.policy/calico-kube-controllers created
```

A variety of resources are created to support pod networking. A daemonset is used to run a Calico-node pod on each node in the cluster. The resources include several custom resources (customresourcedefinition) that extend the Kubernetes API, for example, to support network policies (networkpolicies.crd.projectcalico.org). Many network plugins have a similar installation procedure.

8. Watch the status of the nodes in the cluster:

Copy code

```
watch kubectl get nodes
```



```
NAME           STATUS    ROLES           AGE
ip-10-0-0-11   Ready     control-plane   2m41s
```

With the network plugin initialized, the control-plane node is now **Ready**.

Note: It may take a minute to reach the **Ready** state.

Press *ctrl+c* to stop watching the nodes.

Summary

In this lab step, you used `kubeadm` to initialize a control-plane node. You also initialized a pod network plugin named Calico to fully bring up the control-plane node. The EC2 instance type used in the lab does not have enough CPU capacity to satisfy the CPU resource requests of all of the pods on one instance. The `kube-dns` pod is currently unschedulable due to a lack of CPU resources, although the cluster can operate without it. Once you join a worker node to the cluster in the next lab step, there will be enough CPU capacity for the `kube-dns` pod.

It is worth mentioning that with a single control-plane, there is a single point

ut you should

he procedure is

alancer to distribute

Skip to content Press option + Q to open this menu

VALIDATION CHECKS

1 Checks Start check

○ Created Calico Network Plugin

Check for calico pods to confirm creation.

Kubernetes

Did you like this step?

✕ End Lab

‹ Back

Start check