RX-M Cloud Native Consulting

# Kubernetes

## Lab - Kubernetes cluster expansion using kubeadm

Adding nodes to an existing cluster is an important administrative task. The security aspect of which can be complicated. The `kubeadm` tool provides a `join` command which, in combination with new kublet bootstrap features, can simplify the task, handling all of the certificates and signing needed for a node to join the cluster.

When joining a kubeadm initialized cluster, we need to establish bidirectional trust. This is split into discovery (having the Node trust the Kubernetes Master) and TLS bootstrap (having the Kubernetes Master trust the Node).

There are 2 main schemes for discovery. The first is to use a shared token (which allows the master to trust the new kubelet) along with the IP address of the API server (which allows the kubelet to trust the mstaer). The second is to provide a file - a subset of the standard kubeconfig file. This file can be a local file or downloaded via an HTTPS URL.

Examples of each of the three types of kubeadm join commands:

- `kubeadm join --discovery-token abcdef.1234567890abcdef 1.2.3.4:6443`
- `kubeadm join --discovery-file path/to/file.conf`
- `kubeadm join --discovery-file https://url/file.conf`

If the discovery information is loaded from a URL, HTTPS must be used. Also, in that case the host installed CA bundle is used to verify the connection.

If you use a shared token for discovery, you must also pass the `--discovery-token-ca-cert-hash` flag to validate the public key of the root certificate authority (CA) presented by the Kubernetes Master. This value is available in the output of “kubeadm init”. If you cannot know the CA public key hash ahead of time, you can pass the `--discovery-token-unsafe-skip-ca-verification` flag to disable this verification. This weakens the kubeadm security model since other nodes can potentially impersonate the Kubernetes Master.

The TLS bootstrap mechanism is also driven via a shared token. This is used to temporarily authenticate with the Kubernetes Master to submit a certificate signing request (CSR) for a locally created key pair. By default, kubeadm will set up the Kubernetes Master to automatically approve these signing requests. This token is passed in with the `--tls-bootstrap-token abcdef.1234567890abcdef` flag.

Often times the same token is used for both parts. In this case, the `--token` flag can be used instead of specifying each token individually.

In this lab, we will extend an existing Kubernetes cluster by adding an additional node using the `kubeadm` command set.

### Prerequisites

A single node kubeadm cluster running (see prior labs for instructions).

## 1. Prepare a new Lab VM to add to the cluster

> Your instructor may have provided you with a new VM instance in cloud based labs. If you are running lab VMs in AWS please check to ensure that the Source/Destination Check is set to "disabled" for that instance. In general things are easiest if there are no firewalls between the nodes in the cluster. If you are working on a local machine/laptop you can simply start a new VM as described here: https://github.com/RX-M/classfiles/blob/master/lab-setup.md

Login to the new VM (for example:)

```
laptop$ ssh -i k8s-student.pem ubuntu@<external-ip>

...
ubuntu@ip-172-31-5-10:~$
```

Set the host name for the new VM to *nodeb*:

```
ubuntu@ip-172-31-5-10:~$ sudo hostnamectl set-hostname nodeb

ubuntu@ip-172-31-5-10:~$ cat /etc/hostname

nodeb

ubuntu@ip-172-31-5-10:~$
```

You may need to exit the current shell and open a new shell for your prompt (PS1) to update to the new hostname.

```
ubuntu@ip-172-31-13-140:~$ exit

laptop$

laptop$ ssh -i k8s-student.pem ubuntu@<nodeb-external-ip>

...
ubuntu@nodeb:~$
```

Now discover your IP address (typically eth0 or ens33):

```
ubuntu@nodeb:~$ ip a show eth0

2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc mq state UP group default qlen 1000
    link/ether 02:ab:ae:be:e7:35 brd ff:ff:ff:ff:ff:ff
    inet 172.31.5.10/20 brd 172.31.15.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::ab:aeff:febe:e735/64 scope link
       valid_lft forever preferred_lft forever

ubuntu@nodeb:~$
```

In another terminal, retrieve your master node's IP address:

```
laptop$ ssh -i k8s-student.pem ubuntu@<master-external-ip>

ubuntu@ip-172-31-12-52:~$ ip a s eth0

2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc mq state UP group default qlen 1000
    link/ether 02:d4:97:59:db:43 brd ff:ff:ff:ff:ff:ff
    inet 172.31.12.52/20 brd 172.31.15.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::d4:97ff:fe59:db43/64 scope link
       valid_lft forever preferred_lft forever

ubuntu@ip-172-31-12-52:~$
```

Add the IP address and host name of your new nodeb as well as the current master node to `/etc/hosts`.

> In the example the master node name is listed as ip-172-31-12-52, replace this with the actual hostname of the master node (e.g. "ubuntu", "master", "ip-172-31-12-52", ...).

```
ubuntu@nodeb:~$ sudo cat /etc/hosts

127.0.0.1 localhost
172.31.5.10 nodeb
172.31.12.52 ip-172-31-12-52

# The following lines are desirable for IPv6 capable hosts
```

```
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts

ubuntu@nodeb:~$
```

Repeat these instructions on the master VM, adding ip information for the master and nodeb. In this lab we will refer to the master as ip-172-31-12-52.

On your master node, add *nodeb*'s IP information to the `/etc/hosts` file. Depending on the hypervisor/cloud used your IPs may differ.

```
ubuntu@ip-172-31-12-52:~$ sudo nano /etc/hosts

ubuntu@ip-172-31-12-52:~$ sudo cat /etc/hosts

127.0.0.1 localhost
172.31.5.10 nodeb
172.31.12.52 ip-172-31-12-52

# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts

ubuntu@ip-172-31-12-52:~$
```

Finally, verify that you can reach the internet and both nodes by name with ping from both VMs:

```
ubuntu@nodeb:~$ ping -c 1 k8s.io

PING k8s.io (35.201.71.162) 56(84) bytes of data.
64 bytes from 162.71.201.35.bc.googleusercontent.com (35.201.71.162): icmp_seq=1 ttl=50
time=1.36 ms

--- k8s.io ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.367/1.367/1.367/0.000 ms

ubuntu@nodeb:~$
```

```
ubuntu@nodeb:~$ ping -c 1 ip-172-31-12-52

PING ip-172-31-12-52.us-west-1.compute.internal (172.31.12.52) 56(84) bytes of data.
64 bytes from ip-172-31-12-52 (172.31.12.52): icmp_seq=1 ttl=64 time=0.313 ms

--- ip-172-31-12-52.us-west-1.compute.internal ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.313/0.313/0.313/0.000 ms

ubuntu@nodeb:~$
```

```
ubuntu@ip-172-31-12-52:~$ ping -c 1 nodeb

PING nodeb (172.31.5.10) 56(84) bytes of data.
64 bytes from nodeb (172.31.5.10): icmp_seq=1 ttl=64 time=0.370 ms
```

```
--- nodeb ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.370/0.370/0.370/0.000 ms

ubuntu@ip-172-31-12-52:~$
```

## 2. Install Docker on the Worker Node

Every k8s node will need Docker. In this step we'll install Docker on *nodeb*.

```
ubuntu@nodeb:~$ wget -O - https://get.docker.com | sh

...

ubuntu@nodeb:~$
```

When the system comes back up login and check the version of all parts of the Docker platform with the `docker version` subcommand. You will need to use `sudo` if you did not add the worker's user (ubuntu in this case) to the docker group:

```
ubuntu@nodeb:~$ sudo docker version

Client: Docker Engine - Community
 Version:           19.03.5
 API version:       1.40
 Go version:        go1.12.12
 Git commit:        633a0ea838
 Built:             Wed Nov 13 07:50:12 2019
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          19.03.5
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.12.12
  Git commit:       633a0ea838
  Built:            Wed Nov 13 07:48:43 2019
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.2.10
  GitCommit:        b34a5c8af56e510852c35414db4c1f4fa6172339
 runc:
  Version:          1.0.0-rc8+dev
  GitCommit:        3e425f80a8c931f88e6d94a8c831b9d5aa481657
 docker-init:
  Version:          0.18.0
  GitCommit:        fec3683

ubuntu@nodeb:~$
```

## 3. Install Kubernetes on nodeb

Now we can install the Kubernetes binaries. To do this, we will add the Kubernetes repository to your worker node's apt cache.

Some apt package repos use the aptitude protocol however the Kubernetes packages are served over https, so we need to add the apt https transport:

```
ubuntu@nodeb:~$ sudo apt-get update && sudo apt-get install -y apt-transport-https

Hit:1 http://us-west-1.ec2.archive.ubuntu.com/ubuntu xenial InRelease
Hit:2 http://us-west-1.ec2.archive.ubuntu.com/ubuntu xenial-updates InRelease
Hit:3 http://us-west-1.ec2.archive.ubuntu.com/ubuntu xenial-backports InRelease
Hit:4 https://download.docker.com/linux/ubuntu xenial InRelease
```

```
Hit:5 http://security.ubuntu.com/ubuntu xenial-security InRelease
Reading package lists... Done
Reading package lists... Done
Building dependency tree
Reading state information... Done
apt-transport-https is already the newest version (1.2.32).
0 upgraded, 0 newly installed, 0 to remove and 72 not upgraded.

ubuntu@nodeb:~$
```

Next add the repository key for the Google Cloud packages repository:

```
ubuntu@nodeb:~$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add
-

OK

ubuntu@nodeb:~$
```

Now add a repository list file with an entry for Ubuntu Xenial `apt.kubernetes.io` packages, then update the package indexes to add the Kubernetes packages from `apt.kubernetes.io`:

```
ubuntu@nodeb:~$ echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a
/etc/apt/sources.list.d/kubernetes.list

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

ubuntu@nodeb:~$
```

```
ubuntu@nodeb:~$ sudo apt-get update

Hit:1 http://us-west-1.ec2.archive.ubuntu.com/ubuntu xenial InRelease
Hit:2 http://us-west-1.ec2.archive.ubuntu.com/ubuntu xenial-updates InRelease
Hit:3 http://us-west-1.ec2.archive.ubuntu.com/ubuntu xenial-backports InRelease
Hit:4 https://download.docker.com/linux/ubuntu xenial InRelease
Get:5 https://packages.cloud.google.com/apt kubernetes-xenial InRelease [8,993 B]
Hit:7 http://security.ubuntu.com/ubuntu xenial-security InRelease
Get:6 https://packages.cloud.google.com/apt kubernetes-xenial/main amd64 Packages [33.3 kB]
Fetched 42.3 kB in 0s (83.5 kB/s)
Reading package lists... Done

ubuntu@nodeb:~$
```

Notice the new `kubernetes-xenial` repository above. Now, we can install standard Kubernetes packages.

Use the aptitude package manager to install the kubelet, kubeadm and CNI plugin packages:

```
ubuntu@nodeb:~$ sudo apt-get install -y kubelet kubeadm kubernetes-cni
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  conntrack cri-tools ebtables socat
The following NEW packages will be installed:
  conntrack cri-tools ebtables kubeadm kubelet kubernetes-cni socat
0 upgraded, 8 newly installed, 0 to remove and 72 not upgraded.
Need to get 51.7 MB of archives.
After this operation, 272 MB of additional disk space will be used.

...

Setting up kubernetes-cni (0.7.5-00) ...
Setting up socat (1.7.3.1-1) ...
Setting up kubelet (1.17.2-00) ...
```

```
Setting up kubeadm (1.17.2-00) ...
Processing triggers for ureadahead (0.100.0-19.1) ...
Processing triggers for systemd (229-4ubuntu21.22) ...

ubuntu@nodeb:~$
```

You now have the kubelet, kubeadm and the Kubernetes CNI plugins installed on the worker node.

## 4. kubeadm join

You can now join your worker node with the master and create a multi-node cluster using kubeadm.

The `kubeadm init` command shows instructions for adding new nodes when it completes. Something like this:

> You can now join any number of machines by running the following on each node as root:
>
> kubeadm join 172.31.37.20:6443 --token tvhsyy.ica4fig4tee2gsop --discovery-token-ca-cert-hash sha256:...

Here is a breakdown of the flags following `kubeadm join`:

- `172.31.37.20:6443`: The IP and port of the cluster apiserver to join.
- `--token mxy6qv.rpu2so7531q3xr1d`: A token generated by kubeadm for both discovery-token and tls-bootstrap-token (valid for 24 hours by default)
- `--discovery-token-ca-cert-hash sha256:...`: the sha256 hash of the cluster certificate authority (CA) certificate

The token expires 24 hours after creation. This means that the command above, if run on a worker node, may not work if not run within 24 hours of initializing the master. We can check the validity of a token using the `kubeadm token list` command on your master node. Try it:

```
ubuntu@ip-172-31-12-52:~$ kubeadm token list

TOKEN                     TTL         EXPIRES                USAGES
DESCRIPTION                                            EXTRA GROUPS
fa4nxo.yyesb8ei9irf3itn   23h         2020-01-30T17:26:57Z   authentication,signing   <none>
system:bootstrappers:kubeadm:default-node-token

ubuntu@ip-172-31-12-52:~$
```

> N.B. If running kubeadm token list returns not output, then your cluster has no valid bootstrap tokens. Old tokens are removed from the system after they expire.

If the existing token is expired you can create a new token using the `kubeadm token create` command. Try running this command with the `--dry-run` switch to test the command without actually creating a token:

```
ubuntu@ip-172-31-12-52:~$ kubeadm token create --dry-run

W0129 17:23:31.549744   27212 validation.go:28] Cannot validate kube-proxy config - no validator
is available
W0129 17:23:31.549898   27212 validation.go:28] Cannot validate kubelet config - no validator is
available
[dryrun] Would perform action GET on resource "secrets" in API group "core/v1"
[dryrun] Resource name: "bootstrap-token-dyh1bt"
[dryrun] The GET request didn't yield any result, the API Server returned a NotFound error.
[dryrun] Would perform action CREATE on resource "secrets" in API group "core/v1"
[dryrun] Attached object:
        apiVersion: v1
        data:
          auth-extra-groups: c3lzdGVtOmJvb3RzdHJhcHBlcnM6a3ViZWFkbTpkZWZhdWx0LW5vZGUtdG9rZW4=
          expiration: MjAyMC0wMS0zMFQxNzoyMzozMVo=
          token-id: ZHloMWJ0
          token-secret: NzE1d3F2b2xpd2kwMHdhcw==
          usage-bootstrap-authentication: dHJ1ZQ==
          usage-bootstrap-signing: dHJ1ZQ==
```

```
        kind: Secret
        metadata:
          creationTimestamp: null
          name: bootstrap-token-dyh1bt
          namespace: kube-system
        type: bootstrap.kubernetes.io/token
dyh1bt.715wqvoliwi00was

ubuntu@ip-172-31-12-52:~$
```

As you can see, this command will create a new *Secret* in the kube-system namespace, named `bootstrap-token-<your token id>` containing all the information needed to establish trust between a worker node and a master node. The bootstrap token is also assigned to the system.bootstrappers group, which provides the bearer all the required permissions to connect to the API server, request a new certificate, and retrieve the kubelet configuration.

Let's try to create our own join command, in line with what was generated by `kubeadm init`.

First, we'll need to create our own token id. According to the Kubernetes documentation, any name of the following form is acceptable: `“[a-z0-9]{6}.[a-z0-9]{16}“`. Thankfully, `kubeadm token generate` can generate a name for us, try it:

```
ubuntu@ip-172-31-12-52:~$ kubeadm token generate

ez3mwn.xxdsb8ei9irf3itn

ubuntu@ip-172-31-12-52:~$
```

If you prefer to supply your own token name you can, generating the token is optional. If no token name is passed, `kubeadm create token` will generate one. We'll use the token name we generated. Let's do a dry run of the `kubeadm create` command:

```
ubuntu@ip-172-31-12-52:~$ kubeadm token create ez3mwn.xxdsb8ei9irf3itn --dry-run

W0129 17:25:30.880324   27883 validation.go:28] Cannot validate kube-proxy config - no validator
is available
W0129 17:25:30.880384   27883 validation.go:28] Cannot validate kubelet config - no validator is
available
[dryrun] Would perform action GET on resource "secrets" in API group "core/v1"
[dryrun] Resource name: "bootstrap-token-ez3mwn"
[dryrun] The GET request didn't yield any result, the API Server returned a NotFound error.
[dryrun] Would perform action CREATE on resource "secrets" in API group "core/v1"
[dryrun] Attached object:
        apiVersion: v1
        data:
          auth-extra-groups: c3lzdGVtOmJvb3RzdHJhcHBlcnM6a3ViZWFkbTpkZWZhdWx0LW5vZGUtdG9rZW4=
          expiration: MjAyMC0wMS0zMFQxNzoyNTozMFo=
          token-id: ZXozbXdu
          token-secret: eHhkc2I4ZWk5aXJmM2l0bg==
          usage-bootstrap-authentication: dHJ1ZQ==
          usage-bootstrap-signing: dHJ1ZQ==
        kind: Secret
        metadata:
          creationTimestamp: null
          name: bootstrap-token-ez3mwn
          namespace: kube-system
        type: bootstrap.kubernetes.io/token
ez3mwn.xxdsb8ei9irf3itn

ubuntu@ip-172-31-12-52:~$
```

> The `Cannot validate kube-proxy config` and `Cannot validate kubelet config - no validator is available` errors indicate that there is no mechanism installed to validate those configurations. Tools like `kubeval` or `kubeconfiger` would be invoked at this stage to ensure the kubeconfigs are well formed. With our bare install, we are not installing any and this warning can be safely ignored.

Take note of the Secret's metadata fields.

Now let's create the token in the system by removing the --dry-run flag.

```
ubuntu@ip-172-31-12-52:~$ kubeadm token create ez3mwn.xxdsb8ei9irf3itn

W0129 17:26:57.629642   28373 validation.go:28] Cannot validate kube-proxy config - no validator
is available
W0129 17:26:57.629797   28373 validation.go:28] Cannot validate kubelet config - no validator is
available
ez3mwn.xxdsb8ei9irf3itn

ubuntu@ip-172-31-12-52:~$
```

Now we have the `--token ez3mwn.xxdsb8ei9irf3itn` argument for our join command.

If we list the Secrets available in etcd, we will see our token there under the kube-system namespace, just as our earlier dry run showed:

```
ubuntu@ip-172-31-12-52:~$ kubectl get secrets -n=kube-system | grep bootstrap

bootstrap-signer-token-zcsrg                     kubernetes.io/service-account-token   3
45h
bootstrap-token-ez3mwn                           bootstrap.kubernetes.io/token         6
54s

ubuntu@ip-172-31-12-52:~$
```

Next, we need the discovery token ca cert hash. We will use OpenSSL to do that for us, following the example found in the Kubernetes documentation:

```
ubuntu@ip-172-31-12-52:~$ openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt \
| openssl rsa -pubin -outform der 2>/dev/null \
| openssl dgst -sha256 -hex \
| sed 's/^.* //'

3923325f21d4de5ec2deab3f65fc050b938de5804b1f1b38ca182e958b8044fa

ubuntu@ip-172-31-12-52:~$
```

Here is a breakdown of that command:

- `openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt` : Outputs the public key from the x509 certificate in the ca.crt file
- `| openssl rsa -pubin -outform der 2>/dev/null` : Generates an RSA key using the DER structure from the public key
- `| openssl dgst -sha256 -hex` : Creates a sha256 digest of the DER data output as a hex dump
- `sed 's/^.* //'` : Trims out an erroneous prefix data from the hex dump

That gives us the second piece to our kubeadm join command,
`--discovery-token-ca-cert-hash sha256:3923325f21d4de5ec2deab3f65fc050b938de5804b1f1b38ca182e958b8044fa`
. Be sure to append `sha256:` to the beginning of the output hash.

Now we just need to add the network address of the cluster api server. By default the API server is exposed to port `6443` .

```
ubuntu@ip-172-31-12-52:~$ cat /etc/hosts | grep $(hostname)

172.31.12.52 ip-172-31-12-52

ubuntu@ip-172-31-12-52:~$
```

Thus we have our final piece of information, `172.31.12.52:6443` . Our completed join command is now:

```
sudo kubeadm join \
--token ez3mwn.xxdsb8ei9irf3itn \
--discovery-token-ca-cert-hash
sha256:3923325f21d4de5ec2deab3f65fc050b938de5804b1f1b38ca182e958b8044fa \
172.31.12.52:6443
```

Run the completed command on nodeb:

```
ubuntu@nodeb:~$ sudo kubeadm join \
--token ez3mwn.xxdsb8ei9irf3itn \
--discovery-token-ca-cert-hash
sha256:3923325f21d4de5ec2deab3f65fc050b938de5804b1f1b38ca182e958b8044fa \
172.31.12.52:6443

W0129 17:40:42.117905    7371 join.go:346] [preflight] WARNING: JoinControlPane.controlPlane
settings will be ignored when control-plane flag is not set.
[preflight] Running pre-flight checks
        [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-1.17"
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] Starting the kubelet
[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.

ubuntu@nodeb:~$
```

> We can also use `kubeadm token create --print-join-command` to perform the above steps in one go. you can omit a token name (such as nvv0xn...) to generate a completely random token.

Kubeadm has indicated that nodeb has successfully joined our cluster! As suggested, we'll retrieve a listing of nodes from the master. Switch to the master and run:

```
ubuntu@ip-172-31-12-52:~$  kubectl get nodes

NAME              STATUS   ROLES    AGE   VERSION
ip-172-31-12-52   Ready    master   45h   v1.17.2
nodeb             Ready    <none>   88s   v1.17.2

ubuntu@ip-172-31-12-52:~$
```

We can query docker on nodeb and see that nodeb is now hosting some of the containers it needs to function within the cluster:

```
ubuntu@nodeb:~$ sudo docker container ls

CONTAINER ID        IMAGE                  COMMAND                  CREATED              STATUS
PORTS               NAMES
052e3085a6d6        k8s.gcr.io/coredns     "/coredns -conf /etc…"   About a minute ago   Up
About a minute                      k8s_coredns_coredns-6955765f44-9tqtn_kube-system_2e7a8ae6-
48e6-416f-bad6-97766dae7600_0
f8354c790822        k8s.gcr.io/coredns     "/coredns -conf /etc…"   About a minute ago   Up
```

```
About a minute                       k8s_coredns_coredns-6955765f44-qjhcf_kube-system_49a69c73-
dc70-4b2b-a235-94e14a76d437_0
815053bd207e        k8s.gcr.io/pause:3.1    "/pause"                 About a minute ago   Up
About a minute                       k8s_POD_coredns-6955765f44-9tqtn_kube-system_2e7a8ae6-48e6-
416f-bad6-97766dae7600_0
7deb9a01d851        k8s.gcr.io/pause:3.1    "/pause"                 About a minute ago   Up
About a minute                       k8s_POD_coredns-6955765f44-qjhcf_kube-system_49a69c73-dc70-
4b2b-a235-94e14a76d437_0
8570eb4e91a0        weaveworks/weave-npc    "/usr/bin/launch.sh"     About a minute ago   Up
About a minute                       k8s_weave-npc_weave-net-knjq2_kube-system_9c9a5743-281c-
4615-845a-0dc8032d319d_0
268ca4c34f5e        weaveworks/weave-kube   "/home/weave/launch.…"   About a minute ago   Up
About a minute                       k8s_weave_weave-net-knjq2_kube-system_9c9a5743-281c-4615-
845a-0dc8032d319d_0
67d324b1097c        k8s.gcr.io/kube-proxy   "/usr/local/bin/kube…"   About a minute ago   Up
About a minute                       k8s_kube-proxy_kube-proxy-689dt_kube-system_10e4faa6-0970-
45f8-b146-f762052e07d8_0
41c1f5f11f9e        k8s.gcr.io/pause:3.1    "/pause"                 About a minute ago   Up
About a minute                       k8s_POD_weave-net-knjq2_kube-system_9c9a5743-281c-4615-
845a-0dc8032d319d_0
c3ed48814f94        k8s.gcr.io/pause:3.1    "/pause"                 About a minute ago   Up
About a minute                       k8s_POD_kube-proxy-689dt_kube-system_10e4faa6-0970-45f8-
b146-f762052e07d8_0

ubuntu@nodeb:~$
```

## 6 Exploring the newly bootstrapped node

Previously we explored a `kubeadm init` installed master node. Let's take a look at how `kubeadm join` prepares a node to be a worker.

A worker node will typically need docker to run the containers, a kubelet to communicate with the cluster master's api server and any supporting networking components, such as kube-proxy and a CNI pod.

Let's start with the systemd service and see what we get:

```
ubuntu@nodeb:~$ sudo systemctl status kubelet

● kubelet.service - kubelet: The Kubernetes Node Agent
   Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor preset: enabled)
  Drop-In: /etc/systemd/system/kubelet.service.d
           └─10-kubeadm.conf
   Active: active (running) since Wed 2020-01-29 17:40:42 UTC; 3min 42s ago
     Docs: https://kubernetes.io/docs/home/
 Main PID: 7528 (kubelet)
    Tasks: 18
   Memory: 28.2M
      CPU: 5.078s
   CGroup: /system.slice/kubelet.service
           └─7528 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf
--ku

Jan 29 17:40:44 nodeb kubelet[7528]:         For verbose messaging see
aws.Config.CredentialsChainVer
Jan 29 17:40:48 nodeb kubelet[7528]: W0129 17:40:48.428035    7528 cni.go:237] Unable to update
cni c
Jan 29 17:40:48 nodeb kubelet[7528]: E0129 17:40:48.649673    7528 kubelet.go:2183] Container
runtime
Jan 29 17:40:53 nodeb kubelet[7528]: I0129 17:40:53.344205    7528 transport.go:132] certificate
rota
Jan 29 17:40:53 nodeb kubelet[7528]: W0129 17:40:53.428261    7528 cni.go:237] Unable to update
cni c
Jan 29 17:40:53 nodeb kubelet[7528]: E0129 17:40:53.658300    7528 kubelet.go:2183] Container
runtime
Jan 29 17:41:04 nodeb kubelet[7528]: I0129 17:41:04.939611    7528 reconciler.go:209]
operationExecut
Jan 29 17:41:04 nodeb kubelet[7528]: I0129 17:41:04.939667    7528 reconciler.go:209]
operationExecut
Jan 29 17:41:07 nodeb kubelet[7528]: I0129 17:41:07.044383    7528 reconciler.go:209]
```

```
operationExecut
Jan 29 17:41:07 nodeb kubelet[7528]: I0129 17:41:07.044431    7528 reconciler.go:209]
operationExecut

q

ubuntu@nodeb:~$
```

As on your master node, the apt-get install of the kubelet on *nodeb* installs and configures the kubelet as a systemd service. Our command line output is cut off, so let's look at what arguments were passed using some shell-fu:

```
ubuntu@nodeb:~$ ps -fwwp $(pidof kubelet) | sed -e 's/--/\n--/g'

UID        PID  PPID  C STIME TTY          TIME CMD
root      7528     1  1 17:40 ?        00:00:04 /usr/bin/kubelet
--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf
--kubeconfig=/etc/kubernetes/kubelet.conf
--config=/var/lib/kubelet/config.yaml
--cgroup-driver=cgroupfs
--network-plugin=cni
--pod-infra-container-image=k8s.gcr.io/pause:3.1

ubuntu@nodeb:~$
```

The kubelet arguments are virtually identical to the kubelet setup on your master node.

Let's take a look at the certificates installed on nodeb. If you recall, these are stored in both `/etc/kubernetes/pki` and `/var/lib/kubelet`:

```
ubuntu@nodeb:~$ sudo ls -la /var/lib/kubelet/pki

total 20
drwxr-xr-x 2 root root 4096 Jan 29 17:40 .
drwx------ 8 root root 4096 Jan 29 17:40 ..
-rw------- 1 root root 1061 Jan 29 17:40 kubelet-client-2020-01-29-17-40-43.pem
lrwxrwxrwx 1 root root   59 Jan 29 17:40 kubelet-client-current.pem ->
/var/lib/kubelet/pki/kubelet-client-2020-01-29-17-40-43.pem
-rw-r--r-- 1 root root 2144 Jan 29 17:40 kubelet.crt
-rw------- 1 root root 1679 Jan 29 17:40 kubelet.key

ubuntu@nodeb:~$ sudo ls -la /etc/kubernetes/pki

total 12
drwxr-xr-x 2 root root 4096 Jan 29 17:40 .
drwxr-xr-x 4 root root 4096 Jan 29 17:40 ..
-rw-r--r-- 1 root root 1025 Jan 29 17:40 ca.crt

ubuntu@nodeb:~$
```

The worker node pki artifacts include:

- `ca.crt` - which allows the client to verify the master's identity
- `kubelet.crt` - the client public key certificate that the client presents to the master to identify itself - `kubelet.key` - the private key that allows the client to prove it is the node identified in the client.crt
- `kubelet-client-current.pem` - the current token used by the client

One last thing we can look at is the `etc/kubernetes` directory itself:

```
ubuntu@nodeb:~$ sudo ls -l /etc/kubernetes

total 12
-rw------- 1 root root 1853 Jan 29 17:40 kubelet.conf
drwxr-xr-x 2 root root 4096 Jan 29 17:18 manifests
drwxr-xr-x 2 root root 4096 Jan 29 17:40 pki
```

```
ubuntu@nodeb:~$
```

A manifests directory is available. Remember that any pod specs placed into a manifest directory watched a kubelet will result in a pod being created.

If one were to place a valid pod manifest in that directory, it would be created by the kubelet as a static pod, which is solely managed by the kubelet on that node (in this case, *nodeb*).

On your master node, generate a simple pod manifest without running the pod (--dry-run) in yaml (-o yaml):

```
ubuntu@ip-172-31-12-52:~$ kubectl run --generator=run-pod/v1 mypod --image=nginx --port=80 --
dry-run -o yaml

apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: mypod
  name: mypod
spec:
  containers:
  - image: nginx
    name: mypod
    ports:
    - containerPort: 80
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}

ubuntu@ip-172-31-12-52:~$
```

Copy this output from your master node, trim some of the unnecessary lines such as `status`, `resources` and `creationTimestamp`, and create a pod manifest in *nodeb*'s manifest directory.

```
ubuntu@nodeb:~$ sudo vim /etc/kubernetes/manifests/mypod.yaml

ubuntu@nodeb:~$ sudo cat /etc/kubernetes/manifests/mypod.yaml

apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: mypod
  name: mypod
spec:
  containers:
  - image: nginx
    name: mypod
    ports:
    - containerPort: 80
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}

ubuntu@nodeb:~$
```

Now list the pods on the master.

```
ubuntu@ip-172-31-12-52:~$ kubectl get pods -o wide

NAME          READY   STATUS    RESTARTS   AGE   IP          NODE    NOMINATED NODE   READINESS
```

```
GATES
mypod-nodeb   1/1     Running   0          38s   10.44.0.3   nodeb   <none>           <none>

ubuntu@ip-172-31-12-52:~$
```

The output display's the static pod even though the API server has no way to control it. The kubelet reports the pod but it was not initiated by the apiserver. We will not be able to use `kubectl delete pod` to remove this pod. We need to remove the pod manifest from `/etc/kubernetes/manifests` to eliminate the pod. Try it:

```
ubuntu@nodeb:~$ sudo rm -f /etc/kubernetes/manifests/mypod.yaml

ubuntu@nodeb:~$
```

```
ubuntu@ip-172-31-12-52:~$ kubectl get pods -o wide

No resources found in default namespace.

ubuntu@ip-172-31-12-52:~$
```

## 7. Test the expanded cluster

On the master, list the pods in all namespaces:

```
ubuntu@ip-172-31-12-52:~$ kubectl get pods --all-namespaces -o wide

NAMESPACE     NAME                                       READY   STATUS    RESTARTS   AGE   IP
NODE              NOMINATED NODE   READINESS GATES
kube-system   coredns-6955765f44-9tqtn                   1/1     Running   0          56m
10.44.0.2      nodeb             <none>           <none>
kube-system   coredns-6955765f44-qjhcf                   1/1     Running   0          56m
10.44.0.1      nodeb             <none>           <none>
kube-system   etcd-ip-172-31-12-52                       1/1     Running   0          18h
172.31.12.52   ip-172-31-12-52   <none>           <none>
kube-system   kube-apiserver-ip-172-31-12-52             1/1     Running   0          18h
172.31.12.52   ip-172-31-12-52   <none>           <none>
kube-system   kube-controller-manager-ip-172-31-12-52    1/1     Running   0          18h
172.31.12.52   ip-172-31-12-52   <none>           <none>
kube-system   kube-proxy-689dt                           1/1     Running   0          18m
172.31.5.10    nodeb             <none>           <none>
kube-system   kube-proxy-bpn7j                           1/1     Running   0          18h
172.31.12.52   ip-172-31-12-52   <none>           <none>
kube-system   kube-scheduler-ip-172-31-12-52             1/1     Running   0          18h
172.31.12.52   ip-172-31-12-52   <none>           <none>
kube-system   weave-net-9pntv                            2/2     Running   0          46h
172.31.12.52   ip-172-31-12-52   <none>           <none>
kube-system   weave-net-knjq2                            2/2     Running   0          18m
172.31.5.10    nodeb             <none>           <none>

ubuntu@ip-172-31-12-52:~$
```

The `-o wide` flag shows a number of additional useful columns including the node that the pod is running on. We can see a couple of system pods running on nodeb, the kube-proxy and the weave-net CNI.

Master nodes typically have the "master taint" which keep normal work load pods from running on the master. You may have disabled this taint in order to run test pods. Let's reenable it to see how the taint affects new pod placement. Add the master taint to the master node:

```
ubuntu@ip-172-31-12-52:~$ kubectl taint nodes $(hostname) node-
role.kubernetes.io/master=iso:NoSchedule

ip-172-31-12-52 tainted
```

```
ubuntu@ip-172-31-12-52:~$
```

> N.B. If you did not remove the master node taint when you set up the cluster, kubectl will generate the following error: `error: Node ip-172-31-12-52 already has node-role.kubernetes.io/master taint(s) with same effect(s) and --overwrite is false`. This can be safely ignored.

Let's run a small nginx deployment to test the scheduling with the master tainted:

```
ubuntu@ip-172-31-12-52:~$ kubectl create deploy my-nginx --image=nginx:1.11

deployment.apps/my-nginx created

ubuntu@ip-172-31-12-52:~$
```

If the master node taint, which prevents pods from being scheduled to run on the master node, has not been cleared from our master node, then all the deployed pods will go to nodeb. Let's list the pod distribution again:

```
ubuntu@ip-172-31-12-52:~$ kubectl get pods -o wide

NAME                        READY   STATUS    RESTARTS   AGE   IP          NODE    NOMINATED
NODE   READINESS GATES
my-nginx-5b7bb8ff4f-5mvmk   1/1     Running   0          74s   10.44.0.3   nodeb   <none>
<none>

ubuntu@ip-172-31-12-52:~$
```

Just as expected. Let's clear that master node taint, and then scale up the deployment twofold.

First clear the taint on the master:

```
ubuntu@ip-172-31-12-52:~$ kubectl taint nodes $(hostname) node-role.kubernetes.io/master-

node/ip-172-31-12-52 untainted

ubuntu@ip-172-31-12-52:~$
```

Now scale up:

```
ubuntu@ip-172-31-12-52:~$ kubectl scale deploy/my-nginx --replicas=2

deployment.apps/my-nginx scaled

ubuntu@ip-172-31-12-52:~$
```

Now let's check our pod distribution.

```
ubuntu@ip-172-31-12-52:~$ kubectl get pods -o wide

NAME                        READY   STATUS    RESTARTS   AGE    IP          NODE
NOMINATED NODE   READINESS GATES
my-nginx-5b7bb8ff4f-5mvmk   1/1     Running   0          2m3s   10.44.0.3   nodeb
<none>           <none>
my-nginx-5b7bb8ff4f-np4qk   1/1     Running   0          12s    10.32.0.2   ip-172-31-12-52
<none>           <none>

ubuntu@ip-172-31-12-52:~$
```

With the master tain removed pods again are scheduled across all nodes.

## OPTIONAL: Removing and rejoining a cluster

If desired, we can take any worker node out of the cluster using `kubeadm`. First, let's remove our deployment, ensuring that all our pods are removed from the cluster:

```
ubuntu@ip-172-31-12-52:~$ kubectl delete deploy/my-nginx

deployment.apps "my-nginx" deleted

ubuntu@ip-172-31-12-52:~$
```

Check to ensure the pods are removed:

```
ubuntu@ip-172-31-12-52:~$ kubectl get pods -o wide

No resources found in default namespace.

ubuntu@ip-172-31-12-52:~$
```

Next, we need to inform the cluster that we are removing *nodeb*. First, we will to force any remaining pods off of the node using `kubectl drain`:

```
ubuntu@ip-172-31-12-52:~$ kubectl drain nodeb

node/nodeb cordoned
node/nodeb drained

ubuntu@ip-172-31-12-52:~$
```

Any

Now, we can remove the nodeb api object from etcd:

```
ubuntu@ip-172-31-12-52:~$ kubectl delete node nodeb

node "nodeb" deleted

ubuntu@ip-172-31-12-52:~$ kubectl get nodes

NAME    STATUS   ROLES    AGE    VERSION
ip-172-31-12-52   Ready    master   106m   v1.12.1

ubuntu@ip-172-31-12-52:~$
```

*nodeb* is no longer a part of our cluster. Let's query docker and see what's running on node b:

```
ubuntu@nodeb:~$ sudo docker container ls

CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS
PORTS               NAMES
052e3085a6d6        k8s.gcr.io/coredns     "/coredns -conf /etc…"   25 minutes ago      Up 25
minutes                          k8s_coredns_coredns-6955765f44-9tqtn_kube-system_2e7a8ae6-
48e6-416f-bad6-97766dae7600_0
f8354c790822        k8s.gcr.io/coredns     "/coredns -conf /etc…"   25 minutes ago      Up 25
minutes                          k8s_coredns_coredns-6955765f44-qjhcf_kube-system_49a69c73-
dc70-4b2b-a235-94e14a76d437_0
815053bd207e        k8s.gcr.io/pause:3.1   "/pause"                 25 minutes ago      Up 25
minutes                          k8s_POD_coredns-6955765f44-9tqtn_kube-system_2e7a8ae6-48e6-
416f-bad6-97766dae7600_0
7deb9a01d851        k8s.gcr.io/pause:3.1   "/pause"                 25 minutes ago      Up 25
minutes                          k8s_POD_coredns-6955765f44-qjhcf_kube-system_49a69c73-dc70-
4b2b-a235-94e14a76d437_0
```

```
8570eb4e91a0        weaveworks/weave-npc    "/usr/bin/launch.sh"     25 minutes ago      Up 25
minutes                         k8s_weave-npc_weave-net-knjq2_kube-system_9c9a5743-281c-4615-
845a-0dc8032d319d_0
268ca4c34f5e        weaveworks/weave-kube   "/home/weave/launch.…"   25 minutes ago      Up 25
minutes                         k8s_weave_weave-net-knjq2_kube-system_9c9a5743-281c-4615-845a-
0dc8032d319d_0
67d324b1097c        k8s.gcr.io/kube-proxy   "/usr/local/bin/kube…"   25 minutes ago      Up 25
minutes                         k8s_kube-proxy_kube-proxy-689dt_kube-system_10e4faa6-0970-
45f8-b146-f762052e07d8_0
41c1f5f11f9e        k8s.gcr.io/pause:3.1    "/pause"                 25 minutes ago      Up 25
minutes                         k8s_POD_weave-net-knjq2_kube-system_9c9a5743-281c-4615-845a-
0dc8032d319d_0
c3ed48814f94        k8s.gcr.io/pause:3.1    "/pause"                 25 minutes ago      Up 25
minutes                         k8s_POD_kube-proxy-689dt_kube-system_10e4faa6-0970-45f8-b146-
f762052e07d8_0

ubuntu@nodeb:~$
```

The kubelet on the removed node needs some time to acknowledge its removal. Running the `docker container ls` command on nodeb immediately or shortly after removing it may show the containers running. Eventually, the kubelet will acknowledge its removal and tell the container runtime to delete its running containers:

```
ubuntu@nodeb:~$ sudo docker container ls

CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES

ubuntu@nodeb:~$
```

Kubeadm and the other Kubernetes components are still installed on the machine; let's try to rejoin the cluster:

```
ubuntu@nodeb:~$ sudo kubeadm join \
--token ez3mwn.xxdsb8ei9irf3itn \
--discovery-token-ca-cert-hash
sha256:3923325f21d4de5ec2deab3f65fc050b938de5804b1f1b38ca182e958b8044fa \
172.31.12.52:6443

W0129 18:14:40.089193   21528 join.go:346] [preflight] WARNING: JoinControlPane.controlPlane
settings will be ignored when control-plane flag is not set.
[preflight] Running pre-flight checks
        [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/
error execution phase preflight: [preflight] Some fatal errors occurred:
        [ERROR FileAvailable--etc-kubernetes-kubelet.conf]: /etc/kubernetes/kubelet.conf already
exists
        [ERROR Port-10250]: Port 10250 is in use
        [ERROR FileAvailable--etc-kubernetes-pki-ca.crt]: /etc/kubernetes/pki/ca.crt already
exists
[preflight] If you know what you are doing, you can make a check non-fatal with `--ignore-
preflight-errors=...`
To see the stack trace of this error execute with --v=5 or higher

ubuntu@nodeb:~$
```

It fails at the preflight checks due to existing resources. If we wish to reuse this node at any time, we will need to remove and reverse any changes performed by `kubeadm join`. This is done using `kubeadm reset`:

```
ubuntu@nodeb:~$ sudo kubeadm reset

[reset] WARNING: Changes made to this host by 'kubeadm init' or 'kubeadm join' will be reverted.
[reset] Are you sure you want to proceed? [y/N]: y
[preflight] Running pre-flight checks
W0129 18:15:16.108210   21806 removeetcdmember.go:79] [reset] No kubeadm config, using etcd pod
spec to get data directory
```

```
[reset] No etcd config found. Assuming external etcd
[reset] Please, manually reset etcd to prevent further issues
[reset] Stopping the kubelet service
[reset] Unmounting mounted directories in "/var/lib/kubelet"
[reset] Deleting contents of config directories: [/etc/kubernetes/manifests /etc/kubernetes/pki]
[reset] Deleting files: [/etc/kubernetes/admin.conf /etc/kubernetes/kubelet.conf
/etc/kubernetes/bootstrap-kubelet.conf /etc/kubernetes/controller-manager.conf
/etc/kubernetes/scheduler.conf]
[reset] Deleting contents of stateful directories: [/var/lib/kubelet /var/lib/dockershim
/var/run/kubernetes /var/lib/cni]

The reset process does not clean CNI configuration. To do so, you must remove /etc/cni/net.d

The reset process does not reset or clean up iptables rules or IPVS tables.
If you wish to reset iptables, you must do so manually by using the "iptables" command.

If your cluster was setup to utilize IPVS, run ipvsadm --clear (or similar)
to reset your system's IPVS tables.

The reset process does not clean your kubeconfig files and you must remove them manually.
Please, check the contents of the $HOME/.kube/config file.

ubuntu@nodeb:~$
```

Now that we've reset the node to a pre-kubeadm state, let's try to rejoin the cluster:

```
ubuntu@nodeb:~$ sudo kubeadm join \
--token ez3mwn.xxdsb8ei9irf3itn \
--discovery-token-ca-cert-hash
sha256:3923325f21d4de5ec2deab3f65fc050b938de5804b1f1b38ca182e958b8044fa \
172.31.12.52:6443

W0129 18:15:51.438533   21847 join.go:346] [preflight] WARNING: JoinControlPane.controlPlane
settings will be ignored when control-plane flag is not set.
[preflight] Running pre-flight checks
        [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-1.17"
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] Starting the kubelet
[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.

ubuntu@nodeb:~$
```

We're back!

## OPTIONAL: Retainting the master node

Now that we have a worker node and have confirmed that the scheduler is able to distribute workloads to it (or them), you can optionally restore the taint to your master node.

```
ubuntu@ip-172-31-12-52:~$ kubectl taint nodes $(hostname) node-
role.kubernetes.io/master="":NoSchedule
```

```
node/ip-172-31-12-52 tainted

ubuntu@ip-172-31-12-52:~$
```

Congratulations, you have completed the lab!