RX-M Cloud Native Consulting

Kubernetes

Lab - Kubernetes Cluster Expansion - TLS Bootstrapping

Adding nodes to an existing cluster is an important administrative task.

In this lab, we will manually extend an existing Kubernetes cluster created by kubeadm by adding an additional node using just the kubelet binaries.

Prerequisites

You should already have a single node kubeadm cluster running.

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-172-31-13-140:~$
```

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

```
ubuntu@ip-172-31-9-0:~$ sudo hostnamectl set-hostname nodeb

ubuntu@ip-172-31-9-0:~$ cat /etc/hostname

nodeb

ubuntu@ip-172-31-9-0:~$
```

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 pfifo_fast state UP group default qlen
1000
```

```
    link/ether 02:c5:be:80:6e:65 brd ff:ff:ff:ff:ff:ff
    inet 172.31.8.161/20 brd 172.31.15.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::c5:beff:fe80:6e65/64 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@nodea:~$ 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@nodea:~$
```

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

> In the examples the master node name is listed as nodea, replace this with the actual hostname of the master node (e.g. "ubuntu", "master", "nodea", ...).

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

ubuntu@nodeb:~$ cat /etc/hosts

127.0.0.1 localhost
172.31.15.82 nodea
172.31.8.161 nodeb

# 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.

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

```
ubuntu@nodea:~$ sudo nano /etc/hosts

ubuntu@nodea:~$ cat /etc/hosts

127.0.0.1 localhost
172.31.15.82 nodea
172.31.8.161 nodeb

# 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@nodea:~$
```

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=1.14 ms

--- k8s.io ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.145/1.145/1.145/0.000 ms

ubuntu@nodeb:~$
```

```
ubuntu@nodeb:~$ ping -c 1 nodea

PING nodea (172.31.15.82) 56(84) bytes of data.
64 bytes from nodea (172.31.15.82): icmp_seq=1 ttl=64 time=0.686 ms

--- nodea ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.686/0.686/0.686/0.000 ms

ubuntu@nodeb:~$
```

```
ubuntu@nodea:~$ ping -c 1 nodeb

PING nodeb (172.31.8.161) 56(84) bytes of data.
64 bytes from nodeb (172.31.8.161): icmp_seq=1 ttl=64 time=0.510 ms

--- nodeb ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.510/0.510/0.510/0.000 ms

ubuntu@nodea:~$
```

2. Install Docker on nodeb

Every Kubernetes nodes will need a container runtime, so in this lab we will use Docker. In this step we will install Docker on *nodeb* using the get.docker.com shell script:

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

...

ubuntu@nodeb:~$
```

Check the version of all parts of the Docker platform with the `docker version` subcommand. You will need to use `sudo` since you did not add `ubuntu` to the `docker` user 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 a Kubelet on nodeb

Any worker node in a Kubernetes cluster runs a kubelet process. The kubelet acts as the node agent, running containers through calls to the Docker daemon on the local node and executing instructions from the cluster API server to realize the desired state.

First we need to add the Google cloud packages repo key so that we can install packages hosted by Google:

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

OK
```

Next, add a repository list file with an entry for Ubuntu Xenial `apt.kubernetes.io packages`. The following command copies the repo url into the `kubernetes.list` file:

```
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:~$
```

Update the package indexes to add the Kubernetes packages from apt.kubernetes.io. This makes the Kubernetes binaries available through the Ubuntu aptitude package manager:

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

Hit:1 http://us-west-2.ec2.archive.ubuntu.com/ubuntu xenial InRelease
Hit:2 http://us-west-2.ec2.archive.ubuntu.com/ubuntu xenial-updates InRelease
Hit:3 http://us-west-2.ec2.archive.ubuntu.com/ubuntu xenial-backports InRelease
Hit:4 https://download.docker.com/linux/ubuntu xenial InRelease
Hit:6 http://security.ubuntu.com/ubuntu xenial-security InRelease
Get:5 https://packages.cloud.google.com/apt kubernetes-xenial InRelease [8,993 B]
Get:7 https://packages.cloud.google.com/apt kubernetes-xenial/main amd64 Packages [33.3 kB]
Fetched 42.3 kB in 0s (51.9 kB/s)
Reading package lists... Done
```

Finally, we install the kubelet and the CNI package:

```
ubuntu@nodeb:~$ sudo apt install kubelet kubernetes-cni
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  conntrack ebtables socat
The following NEW packages will be installed:
  conntrack ebtables kubelet kubernetes-cni socat
0 upgraded, 5 newly installed, 0 to remove and 68 not upgraded.
Need to get 26.1 MB of archives.
After this operation, 163 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://us-west-2.ec2.archive.ubuntu.com/ubuntu xenial/main amd64 conntrack amd64 1:1.4.3-3
[27.3 kB]
Get:2 http://us-west-2.ec2.archive.ubuntu.com/ubuntu xenial-updates/main amd64 ebtables amd64
2.0.10.4-3.4ubuntu2.16.04.2 [79.9 kB]
Get:3 http://us-west-2.ec2.archive.ubuntu.com/ubuntu xenial/universe amd64 socat amd64 1.7.3.1-1
[321 kB]
Get:4 https://packages.cloud.google.com/apt kubernetes-xenial/main amd64 kubernetes-cni amd64
0.7.5-00 [6,473 kB]
Get:5 https://packages.cloud.google.com/apt kubernetes-xenial/main amd64 kubelet amd64 1.17.2-00
[19.2 MB]
Fetched 26.1 MB in 1s (16.5 MB/s)
Selecting previously unselected package conntrack.
(Reading database ... 51613 files and directories currently installed.)
Preparing to unpack .../conntrack_1%3a1.4.3-3_amd64.deb ...
Unpacking conntrack (1:1.4.3-3) ...
Selecting previously unselected package ebtables.
Preparing to unpack .../ebtables_2.0.10.4-3.4ubuntu2.16.04.2_amd64.deb ...
Unpacking ebtables (2.0.10.4-3.4ubuntu2.16.04.2) ...
Selecting previously unselected package kubernetes-cni.
Preparing to unpack .../kubernetes-cni_0.7.5-00_amd64.deb ...
Unpacking kubernetes-cni (0.7.5-00) ...
Selecting previously unselected package socat.
Preparing to unpack .../socat_1.7.3.1-1_amd64.deb ...
Unpacking socat (1.7.3.1-1) ...
Selecting previously unselected package kubelet.
Preparing to unpack .../kubelet_1.17.2-00_amd64.deb ...
Unpacking kubelet (1.17.2-00) ...
Processing triggers for man-db (2.7.5-1) ...
Processing triggers for ureadahead (0.100.0-19.1) ...
Processing triggers for systemd (229-4ubuntu21.22) ...
Setting up conntrack (1:1.4.3-3) ...
Setting up ebtables (2.0.10.4-3.4ubuntu2.16.04.2) ...
update-rc.d: warning: start and stop actions are no longer supported; falling back to defaults
Setting up kubernetes-cni (0.7.5-00) ...
Setting up socat (1.7.3.1-1) ...
Setting up kubelet (1.17.2-00) ...
Processing triggers for ureadahead (0.100.0-19.1) ...
Processing triggers for systemd (229-4ubuntu21.22) ...
```

This installs the kubelet as a systemd service on your new node. We do need to reconfigure it, so for now stop it:

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

ubuntu@nodeb:~$
```

4. Prepare certificates for the new worker node

Production Kubernetes systems generally use TLS to protect all intra-cluster communications. The API Server authenticates nodes using TLS certificates. When you use an automated process like `kubeadm join`, the node certificates are automatically generated for and distributed to the new worker nodes.

Each worker in your cluster needs the cluster's certificate authority (CA) certificate, `ca.crt`, which your kubelet will authenticate responses from the cluster.

We will recreate the ca.crt from your master node on the worker using nano. First, use `cat` on your master node to get the current

CA certificate:

```
ubuntu@nodea:~$ sudo cat /etc/kubernetes/pki/ca.crt

-----BEGIN CERTIFICATE-----
MIICyDCCAbCgAwIBAgIBADANBgkqhkiG9w0BAQsFADAVMRMwEQYDVQQDEwprdWJl
cm5ldGVzMB4XDTIwMDEyMjIyNDk1M1oXDTMwMDExOTIyNDk1M1owFTETMBEGA1UE
AxMKa3ViZXJuZXRlczCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAMd3
LRL7mU4puZ9ODU7D21AcZB1eGk/zd63fgeArYOe8h9AJDHGFp+S+nc5R0CVs5u9V
zSb3V9kolInkiKjsZKfs6QE+hD1C1iWo/02l8ncvzmdM1FyFt701nxziIx4y/Lo/
ggxBpPrdS1XwtgH7MKfZqCxv7F+5WX7rM14GTKr1qXyRv/PdbBPonWoLuO5pPMsV
DJ/MPXTFnqj1tef4BblwD31v9V5pulIKC29boGYd9Q4Vw7A26/lMYrVFPuaz658H
5x5IYn/x4JYFFQowN2bw8FRg3pLsoRMK4vjF7CwgiT0r7TWV0A6NcRhkGkgskSVK
X90AxFDIoMLWgIP+sm8CAwEAAaMjMCEwDgYDVR0PAQH/BAQDAgKkMA8GA1UdEwEB
/wQFMAMBAf8wDQYJKoZIhvcNAQELBQADggEBAIM40J2yVbVMEinpC61Eiiq8NRn5
uXdox+V4oiafmBkssvp/yCN+fuvRih9IgSBzZPLOBitqsVnY5DQ+rWV7lokopBl9
Ph17x4d5MB82skxdEZGglB682mzeIECpmMFtJJgpoVvux78zebj/fBq25cmpBASR
JhvLyDojOtnUDWjami+crVGKhCT56LHImZMFK4DKSdIBa5hEXMPRHmXoGpCFAYSZ
z3vmSs7/59IkLKMQd28E6Qvp7VDxtLqypk94wZ/KYxJZ7183GL0p5KyzjDUmqzr7
Lhsb4anol61tKpdDRDDt5a+B1h553Tdrn2/Qx8cs8AXi/fiYMq7HvbLRQkM=
-----END CERTIFICATE-----

ubuntu@nodea:~$
```

Copy those contents from nodea to a file in the same location on nodeb. You may need to create the `/etc/kubernetes/pki` directory first:

```
ubuntu@nodeb:~$ sudo mkdir /etc/kubernetes/pki

ubuntu@nodeb:~$ sudo nano /etc/kubernetes/pki/ca.crt

ubuntu@nodeb:~$ sudo cat /etc/kubernetes/pki/ca.crt

-----BEGIN CERTIFICATE-----
MIICyDCCAbCgAwIBAgIBADANBgkqhkiG9w0BAQsFADAVMRMwEQYDVQQDEwprdWJl
cm5ldGVzMB4XDTIwMDEyMjIyNDk1M1oXDTMwMDExOTIyNDk1M1owFTETMBEGA1UE
AxMKa3ViZXJuZXRlczCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAMd3
LRL7mU4puZ9ODU7D21AcZB1eGk/zd63fgeArYOe8h9AJDHGFp+S+nc5R0CVs5u9V
zSb3V9kolInkiKjsZKfs6QE+hD1C1iWo/02l8ncvzmdM1FyFt701nxziIx4y/Lo/
ggxBpPrdS1XwtgH7MKfZqCxv7F+5WX7rM14GTKr1qXyRv/PdbBPonWoLuO5pPMsV
DJ/MPXTFnqj1tef4BblwD31v9V5pulIKC29boGYd9Q4Vw7A26/lMYrVFPuaz658H
5x5IYn/x4JYFFQowN2bw8FRg3pLsoRMK4vjF7CwgiT0r7TWV0A6NcRhkGkgskSVK
X90AxFDIoMLWgIP+sm8CAwEAAaMjMCEwDgYDVR0PAQH/BAQDAgKkMA8GA1UdEwEB
/wQFMAMBAf8wDQYJKoZIhvcNAQELBQADggEBAIM40J2yVbVMEinpC61Eiiq8NRn5
uXdox+V4oiafmBkssvp/yCN+fuvRih9IgSBzZPLOBitqsVnY5DQ+rWV7lokopBl9
Ph17x4d5MB82skxdEZGglB682mzeIECpmMFtJJgpoVvux78zebj/fBq25cmpBASR
JhvLyDojOtnUDWjami+crVGKhCT56LHImZMFK4DKSdIBa5hEXMPRHmXoGpCFAYSZ
z3vmSs7/59IkLKMQd28E6Qvp7VDxtLqypk94wZ/KYxJZ7183GL0p5KyzjDUmqzr7
Lhsb4anol61tKpdDRDDt5a+B1h553Tdrn2/Qx8cs8AXi/fiYMq7HvbLRQkM=
-----END CERTIFICATE-----

ubuntu@nodeb:~$
```

With the CA certificate copied, you can now continue setting up the kubelet on your new worker.

5. Prepare bootstrap-kubelet.conf

The next step is to prepare the `bootstrap-kubelet.conf` which the kubelet will use to try to register itself with the cluster.

The `bootstrap-kubelet.conf` file provides the URL for the server and bootstrap credentials. These credentials come in the form of tokens. A bootstrap token, if present in your cluster through provisioning (using tools like kubeadm) or a static token file (which you may find defined in a file by the `--token-auth-file` flag on the kube-apiserver), is stored on the cluster as a secret under the kube-system namespace:

```
ubuntu@nodea:~$ kubectl get secrets -n kube-system
```

```
NAME                                         TYPE                                  DATA
AGE
attachdetach-controller-token-dgpmj          kubernetes.io/service-account-token   3
85m
bootstrap-signer-token-82hlb                 kubernetes.io/service-account-token   3
85m
bootstrap-token-ibpecg                       bootstrap.kubernetes.io/token         7
85m
...

ubuntu@nodea:~$
```

N.B. In Kubeadm clusters like ours, bootstrap tokens are only valid for 24 hours. If your bootstrap token is over 24 hours old, use kubeadm to create a new token with `kubeadm token create`. This command outputs a complete bootstrap token to the terminal and creates a corresponding secret in the cluster.

```
ubuntu@nodea:~$ kubeadm token create

ezgumy.l3p5h3hn4xsd9u2k

ubuntu@nodea:~$
```

The `bootstrap.kubernetes.io/token` type of tokens are associated with the `system:bootstrappers` group. From the kube-apiserver's perspective a bootstrap token is special due to its `Type`, `namespace` and `name`. The cluster's kube-apiserver recognizes it as a special token, and grants any entity authenticating with that token special bootstrap rights.

The main advantage of using bootstrap tokens and the entire TLS bootstrap process is the hands-off approach to certificates. Without these tokens, you will need to manually use tools like `openssl` or `cfssl` to create and sign certificates and distribute them to all nodes in the cluster.

Describe the secret for the `bootstrap.kubernetes.io/token` in your cluster:

```
ubuntu@nodea:~$ kubectl get secret -n kube-system bootstrap-token-ibpecg -o yaml

apiVersion: v1
data:
  auth-extra-groups: c3lzdGVtOmJvb3RzdHJhcHBlcnM6a3ViZWFkbTpkZWZhdWx0LW5vZGUtdG9rZW4=
  description: VGhlIGRlZmF1bHQgYm9vdHN0cmFwIHRva2VuIGdlbmVyYXRlZCBieSAna3ViZWFkbSBpbml0Jy4=
  expiration: MjAyMC0wMS0yM1QyMjo1MDoxM1o=
  token-id: aWJwZWNn
  token-secret: aTBhcmM3aWY0d2o3bmozcA==
  usage-bootstrap-authentication: dHJ1ZQ==
  usage-bootstrap-signing: dHJ1ZQ==
kind: Secret
metadata:
  creationTimestamp: "2020-01-22T22:50:13Z"
  name: bootstrap-token-ibpecg
  namespace: kube-system
  resourceVersion: "178"
  selfLink: /api/v1/namespaces/kube-system/secrets/bootstrap-token-ibpecg
  uid: 668e4dd5-4220-4f01-b58f-80fd1eaa57df
type: bootstrap.kubernetes.io/token

ubuntu@nodea:~$
```

The values of note in this secret are the `token-id` and `token-secret`, which together make up a complete access token. These values are currently encoded in base64 within the secret. Use a combination of `kubectl get`, `-o jsonpath`, and the Linux `base64` tool to decode those values:

```
ubuntu@nodea:~$ export BOOTSTRAPSECRET=bootstrap-token-ibpecg && echo $BOOTSTRAPSECRET
```

```
bootstrap-token-ibpecg

ubuntu@nodea:~$ kubectl get secret -n kube-system $BOOTSTRAPSECRET -o jsonpath='{.data.token-
id}' | base64 --decode ; echo

ibpecg

ubuntu@nodea:~$ kubectl get secret -n kube-system $BOOTSTRAPSECRET -o jsonpath='{.data.token-
secret}' | base64 --decode ; echo

i0arc7if4wj7nj3p

ubuntu@nodea:~$
```

A full token is formatted as the ., so together your complete bootstrap token is `ibpecg.i0arc7if4wj7nj3p`

With the bootstrap token, you can now begin constructing the `bootstrap-kubelet.conf` file. It is formatted in the same way as a regular kubeconfig file. An example of such a file is shown below:

```
apiVersion: v1
kind: Config
clusters:
- cluster:
    certificate-authority: /etc/kubernetes/pki/ca.crt
    server: https://my.server.example.com:6443
  name: bootstrap
contexts:
- context:
    cluster: bootstrap
    user: kubelet-bootstrap
  name: bootstrap
current-context: bootstrap
preferences: {}
users:
- name: kubelet-bootstrap
  user:
    token: 07401b.f395accd246ae52d
```

The first part, clusters, will need the connection information to the API server. Retrieve this from the nodea `kubelet.conf` file under `/etc/kubernetes`:

```
ubuntu@nodea:~$ sudo head /etc/kubernetes/kubelet.conf

apiVersion: v1
clusters:
- cluster:
    certificate-authority-data:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUN5RENDQWJDZ0F3SUJBZ0lCQURBTkJna3Foa2lHOXcwQkFRc0ZBREFW
TVJNd0VRWURWUVFERXdwcmRXSmwKY201bGRHVnpNQjRYRFRJd01ERXlNakl5TkRrMU0xb1hEVE13TURFeE9USXlORGsxTTFv
d0ZURVRNQkVHQTFVRQpBeE1LYTNWaVpYSnVaWFJsY3pDQ0FTSXdEUVlKS29aSWh2Y05BUUVCQlFBRGdnRVBBRENDQVFvQ2dn
RUJBTWQzCkxSTDdtVTRwdVo5T0RVN0QyMUFjWkIxZUdrL3pkNjNmZ2VBcllPZThoOUFKREhHRnArUytuYzVSMENWczV1OVYK
elNiM1Y5a29sSW5raUtqc1pLZnM2UUUraEQxQzFpV28vMDJsOG5jdnptZE0xRnlGdDcwMW54emlJeDR5L0xvLwpnZ3hCcFBy
ZFMxWHd0Z0g3TUtmWnFDeHY3Ris1V1g3ck0xNEdUS3IxcVh5UnYvUGRiQlBvbldvTHVPNXBQTXNWCkRKL01QWFRGbnFqMXRl
ZjRCYmx3RDMxdjlWNXB1bElLQzI5Ym9HWWQ5UTRWdzdBMjYvbE1ZclZGUHVhejY1OEgKNXg1SVluL3g0SllGRlFvd04yYnc4
RlJnM3BMc29STUs0dmpGN0N3Z2lUMHI3VFdWMEE2TmNSaGtHa2dza1NWSwpYOTBBeEZESW9NTFdnSVArc204Q0F3RUFBYU1q
TUNFd0RnWURWUjBQQVFIL0JBUURBZ0trTUE4R0ExVWRFd0VCCi93UUZNQU1CQWY4d0RRWUpLb1pJaHZjTkFRUxCUUFEZ2dF
QkFJTTQwSjJ5VmJWTUVpbnBDNjFFaWlxOE5SbjUKdVhkb3grVjRvaWFmbUJrc3N2cC95Q04rZnV2UmloOUlnU0J6WlBMT0Jp
dHFzVm5ZNURRK3JXVjdsb2tvcEJsOQpQaDE3eDRkNU1CODJza3hkRVpHZ2xCNjgybXplSUVDcG1NRnRKSmdwb1Z2dXg3OHpl
YmovZkJxMjVjbXBCQVNSCkpodkx5RG9qT3RuVURXamFtaStjclZHS2hDVDU2TEhJbVpNRks0REtTZElCYTVoRVhNUFJIbVhv
R3BDRkFZU1oKejN2bVNzNy81OUlrTEtNUWQyOEU2UXZwN1ZEeHRMcXlwazk0d1ovS1l4Slo3MTgzR0wwcDVLeXpqRFVtcXpy
NwpMaHNiNGFub2w2MXRLcGREUkREdDVhK0IxaDU1M1Rkcm4yL1F4OGNzOEFYaS9maVlNcTdIdmJMUlFrTT0KLS0tLS1FTkQg
Q0VSVElGSUNBVEUtLS0tLQo=
    server: https://172.31.47.36:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: system:node:nodea
```

```
ubuntu@nodea:~$
```

`kubectl` is able to construct kubeconfigs for us with a few imperative commands. First, we'll add the API server connection and certificate authority (the path to the cluster's CA certificate you recreated on the worker node earlier) information to the file using `kubectl config set-cluster`:

```
ubuntu@nodea:~$ kubectl config set-cluster bootstrap \
--kubeconfig=bootstrap-kubelet.conf \
--certificate-authority=/etc/kubernetes/pki/ca.crt \
--server='https://172.31.47.36:6443'

Cluster "bootstrap" set.

ubuntu@nodea:~$
```

We'll write a bootstrap-kubelet.conf file to our home directory for now. This first command sets up a cluster called bootstrap that points to our API server and the cluster's CA cert. This command populates the `clusters:` section of the kubeconfig.

Next, associate the bootstrap token you retrieved earlier to a user called `kubelet-bootstrap`:

```
ubuntu@nodea:~$ kubectl config set-credentials kubelet-bootstrap \
--kubeconfig=bootstrap-kubelet.conf \
--token=ibpecg.i0arc7if4wj7nj3p

User "kubelet-bootstrap" set.

ubuntu@nodea:~$
```

This creates the `users:` section of the kubeconfig, placing the token as the authentication method for to the user `kubelet-bootstrap`.

Now, set a context that associates the `kubelet-bootstrap` user to the cluster connection information:

```
ubuntu@nodea:~$ kubectl config set-context bootstrap \
--kubeconfig=bootstrap-kubelet.conf \
--user=kubelet-bootstrap --cluster=bootstrap

Context "bootstrap" created.

ubuntu@nodea:~$
```

This creates the `contexts:` section of the kubeconfig.

Finally, create the file by switching to the bootstrap context with `kubectl config use-context`:

```
ubuntu@nodea:~$ kubectl config use-context bootstrap \
--kubeconfig=bootstrap-kubelet.conf

Switched to context "bootstrap".

ubuntu@nodea:~$
```

`kubectl config use-context` typically edits the current context of a kubeconfig, but since the file does not exist it actually creates `bootstrap-kubelet.conf` under in your home directory. Use `cat` to view the contents:

```
ubuntu@nodea:~$ cat bootstrap-kubelet.conf

apiVersion: v1
clusters:
- cluster:
    certificate-authority: /etc/kubernetes/pki/ca.crt
```

```
    server: https://172.31.47.36:6443
  name: bootstrap
contexts:
- context:
    cluster: bootstrap
    user: kubelet-bootstrap
  name: bootstrap
current-context: bootstrap
kind: Config
preferences: {}
users:
- name: kubelet-bootstrap
  user:
    token: ibpecg.i0arc7if4wj7nj3p

ubuntu@nodea:~$
```

Recreate this file on nodeb under `/etc/kubernetes`. Remember to use `sudo` since that directory is privileged:

```
ubuntu@nodeb:~$ sudo nano /etc/kubernetes/bootstrap-kubelet.conf

ubuntu@nodeb:~$ sudo cat /etc/kubernetes/bootstrap-kubelet.conf

apiVersion: v1
clusters:
- cluster:
    certificate-authority: /etc/kubernetes/pki/ca.crt
    server: https://172.31.47.36:6443
  name: bootstrap
contexts:
- context:
    cluster: bootstrap
    user: kubelet-bootstrap
  name: bootstrap
current-context: bootstrap
kind: Config
preferences: {}
users:
- name: kubelet-bootstrap
  user:
    token: ibpecg.i0arc7if4wj7nj3p

ubuntu@nodeb:~$
```

Once you have made the changes, the bootstrap-kubelet.conf is now ready. Your new worker node will use the `kubelet-bootstrap` user to initially bootstrap itself into the cluster.

6. Prepare Kubelet config.yaml

The config.yaml is a configuration file that is created when using `kubeadm init` to bootstrap a cluster. This file contains configuration arguments that will influence a kubelet in the cluster. This configuration file is also created as a configmap within the cluster itself.

Switch back to your nodea terminal session. To replicate the kubelet worker node settings, read the kubelet `config.yaml` file from the master node:

```
ubuntu@nodea:~$ sudo cat /var/lib/kubelet/config.yaml

apiVersion: kubelet.config.k8s.io/v1beta1
authentication:
  anonymous:
    enabled: false
  webhook:
    cacheTTL: 0s
    enabled: true
  x509:
    clientCAFile: /etc/kubernetes/pki/ca.crt
```

```
authorization:
  mode: Webhook
  webhook:
    cacheAuthorizedTTL: 0s
    cacheUnauthorizedTTL: 0s
clusterDNS:
- 10.96.0.10
clusterDomain: cluster.local
cpuManagerReconcilePeriod: 0s
evictionPressureTransitionPeriod: 0s
fileCheckFrequency: 0s
healthzBindAddress: 127.0.0.1
healthzPort: 10248
httpCheckFrequency: 0s
imageMinimumGCAge: 0s
kind: KubeletConfiguration
nodeStatusReportFrequency: 0s
nodeStatusUpdateFrequency: 0s
rotateCertificates: true
runtimeRequestTimeout: 0s
staticPodPath: /etc/kubernetes/manifests
streamingConnectionIdleTimeout: 0s
syncFrequency: 0s
volumeStatsAggPeriod: 0s

ubuntu@nodea:~$
```

On nodeb, create a copy of the `config.yaml` file under `/var/lib/kubelet/`. Use `cat` to read the file to see if there are any files whose presence you need to verify:

```
ubuntu@nodeb:~$ sudo nano /var/lib/kubelet/config.yaml

ubuntu@nodeb:~$ sudo cat /var/lib/kubelet/config.yaml

apiVersion: kubelet.config.k8s.io/v1beta1
authentication:
  anonymous:
    enabled: false
  webhook:
    cacheTTL: 0s
    enabled: true
  x509:
    clientCAFile: /etc/kubernetes/pki/ca.crt
authorization:
  mode: Webhook
  webhook:
    cacheAuthorizedTTL: 0s
    cacheUnauthorizedTTL: 0s
clusterDNS:
- 10.96.0.10
clusterDomain: cluster.local
cpuManagerReconcilePeriod: 0s
evictionPressureTransitionPeriod: 0s
fileCheckFrequency: 0s
healthzBindAddress: 127.0.0.1
healthzPort: 10248
httpCheckFrequency: 0s
imageMinimumGCAge: 0s
kind: KubeletConfiguration
nodeStatusReportFrequency: 0s
nodeStatusUpdateFrequency: 0s
rotateCertificates: true
runtimeRequestTimeout: 0s
staticPodPath: /etc/kubernetes/manifests
streamingConnectionIdleTimeout: 0s
syncFrequency: 0s
volumeStatsAggPeriod: 0s

ubuntu@nodeb:~$
```

In this file the cluster certificate authority certificate is expected under `/etc/kubernetes/pki`. You recreated this file earlier when you retrieved the certificates, but use `ls` to check if the file is present:

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

total 4
-rw-r--r-- 1 root root 1025 Jan  7 22:16 ca.crt

ubuntu@nodeb:~$
```

Other than this check, no changes or action are required in this file.

7. Prepare the Kubelet arguments

Before you start the Kubelet, you will need to ensure that it is aware of what kubeconfig to use, what kind of networking the kubelet will use, and provide other switches. In this case, we want to bootstrap the Kubelet with the same flags as our master's kubelet.

On your master node, use `ps -ef | grep kubelet` to see the flags:

```
ubuntu@nodea:~$ ps -ef | grep /usr/bin/kubelet

root      7895     1  1 00:51 ?        00:00:07 /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   11724  5026  0 00:58 pts/0    00:00:00 grep --color=auto /usr/bin/kubelet

ubuntu@nodea:~$
```

The flag of note is `--network-plugin=cni`. This option tells the kubelet which network plugin to invoke for various events in kubelet/pod lifecycle. This flag only works when container runtime is set to `docker`, which your new worker node is using. Omitting this flag will cause the kubelet on this node to assign pods the Docker container IP address.

In Kubeadm bootstrapped clusters, those kubelet flags provided by a separate file. Check the drop-in for the systemd service:

```
ubuntu@nodea:~$ sudo cat /etc/systemd/system/kubelet.service.d/10-kubeadm.conf

# Note: This dropin only works with kubeadm and kubelet v1.11+
[Service]
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-
kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf"
Environment="KUBELET_CONFIG_ARGS=--config=/var/lib/kubelet/config.yaml"
# This is a file that "kubeadm init" and "kubeadm join" generates at runtime, populating the
KUBELET_KUBEADM_ARGS variable dynamically
EnvironmentFile=-/var/lib/kubelet/kubeadm-flags.env
# This is a file that the user can use for overrides of the kubelet args as a last resort.
Preferably, the user should use
# the .NodeRegistration.KubeletExtraArgs object in the configuration files instead.
KUBELET_EXTRA_ARGS should be sourced from this file.
EnvironmentFile=-/etc/default/kubelet
ExecStart=
ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_CONFIG_ARGS $KUBELET_KUBEADM_ARGS
$KUBELET_EXTRA_ARGS

ubuntu@nodea:~$
```

The kubelet flags are stored inside `/var/lib/kubelet/kubeadm-flags.env`. Check the contents of that file:

```
ubuntu@nodea:~$ sudo cat /var/lib/kubelet/kubeadm-flags.env

KUBELET_KUBEADM_ARGS="--cgroup-driver=cgroupfs --network-plugin=cni --pod-infra-container-
image=k8s.gcr.io/pause:3.1"
```

```
ubuntu@nodea:~$
```

When you define `cni` as the network plugin, the kubelet will attempt to run an agent that is responsible for allocating a subnet lease out of a preconfigured address space. The agent binaries are typically installed with the kubernetes-cni package, which is listed as a dependency for the kubelet when it is installed using a package manager. The agent binaries and are found under `/opt/cni/bin`.

In addition to those flags, we need to add the `--rotate-server-certificates` argument to our kubelet.

`--rotate-server-certificates` tells the kubelet to request a serving certificate after bootstrapping its client credentials and to rotate that certificate. The serving certificate is used by optional components like the Kubernetes metrics server to verify the identity of kubelets when scraping metrics. Omitting this flag will allow the node to join the cluster, but the serving certificate will be missing and those other services like the metrics server will be unable to verify the certificate.

Edit the kubelet service file and add the following flags to the ExecStart line:

- `--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf` - declares the bootstrap configuration file
- `--kubeconfig=/etc/kubernetes/kubelet.conf` - declares where the running kubelet configuration file must be written
- `--cgroup-driver=cgroupfs` - This tells what mechanism to use for creating cgroups
- `--network-plugin=cni` - Declares what types of network plugin to use for the kubelet
- `--pod-infra-container-image=k8s.gcr.io/pause:3.1` - Tells the kubelet what to use for the infrastructure containers
- `--rotate-server-certificates` - tells the kubelet to perform server certificate rotation
- `--config=/var/lib/kubelet/config.yaml` - tells the kubelet the location of the configuration file

```
ubuntu@nodeb:~$ sudo nano /lib/systemd/system/kubelet.service

ubuntu@nodeb:~$ sudo cat /lib/systemd/system/kubelet.service

[Unit]
Description=kubelet: The Kubernetes Node Agent
Documentation=https://kubernetes.io/docs/home/

[Service]
ExecStart=/usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --
kubeconfig=/etc/kubernetes/kubelet.conf --cgroup-driver=cgroupfs --network-plugin=cni --pod-
infra-container-image=k8s.gcr.io/pause:3.1 --rotate-server-certificates --
config=/var/lib/kubelet/config.yaml
Restart=always
StartLimitInterval=0
RestartSec=10

[Install]
WantedBy=multi-user.target

ubuntu@nodeb:~$
```

You now have all the files and binaries required to successfully start the kubelet service on your new worker node.

8. Start the kubelet

Make sure the kubelet service is aware of the new files (particularly the new flags) using `sudo systemctl daemon-reload` and start it using `sudo systemctl start kubelet`:

```
ubuntu@nodeb:~$ sudo systemctl daemon-reload

ubuntu@nodeb:~$ sudo systemctl start kubelet

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

Check the status of the service using `sudo systemctl status`:

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

```
● kubelet.service - kubelet: The Kubernetes Node Agent
   Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2020-01-22 23:59:45 UTC; 48s ago
     Docs: https://kubernetes.io/docs/home/
 Main PID: 6492 (kubelet)
    Tasks: 15
   Memory: 23.7M
      CPU: 992ms
   CGroup: /system.slice/kubelet.service
           └─6492 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf
--ku

Jan 23 00:00:00 nodeb kubelet[6492]: W0123 00:00:00.826436    6492 cni.go:237] Unable to update
cni c
Jan 23 00:00:01 nodeb kubelet[6492]: E0123 00:00:01.012630    6492 kubelet.go:2183] Container
runtime
Jan 23 00:00:05 nodeb kubelet[6492]: W0123 00:00:05.826668    6492 cni.go:237] Unable to update
cni c
Jan 23 00:00:06 nodeb kubelet[6492]: E0123 00:00:06.020732    6492 kubelet.go:2183] Container
runtime
Jan 23 00:00:10 nodeb kubelet[6492]: W0123 00:00:10.826946    6492 cni.go:237] Unable to update
cni c
Jan 23 00:00:11 nodeb kubelet[6492]: E0123 00:00:11.028745    6492 kubelet.go:2183] Container
runtime
Jan 23 00:00:15 nodeb kubelet[6492]: W0123 00:00:15.827204    6492 cni.go:237] Unable to update
cni c
Jan 23 00:00:16 nodeb kubelet[6492]: E0123 00:00:16.036813    6492 kubelet.go:2183] Container
runtime
Jan 23 00:00:20 nodeb kubelet[6492]: W0123 00:00:20.827459    6492 cni.go:237] Unable to update
cni c
Jan 23 00:00:21 nodeb kubelet[6492]: E0123 00:00:21.044748    6492 kubelet.go:2183] Container
runtime

ubuntu@nodeb:~$
```

The kubelet is now active, meaning all of the files are in place.

> If your Kubelet is reporting a missing /etc/kuberentes/manifests directory, creating it with `sudo mkdir /etc/kubernetes/manifests` will suppress the warning.

Now, check to see if your node has joined the cluster. Retrieve a listing of nodes from the master. Switch to the master and run:

```
ubuntu@nodea:~$ kubectl get nodes

NAME    STATUS     ROLES    AGE   VERSION
nodea   Ready      master   69m   v1.17.2
nodeb   NotReady   <none>   11s   v1.17.2

ubuntu@nodea:~$
```

Before we can use the node, we need to approve the new node's certificate signing request.

Check the certificatesigningrequests resources available in your cluster using `kubectl get csr`:

```
ubuntu@nodea:~$ kubectl get csr

NAME        AGE   REQUESTOR                 CONDITION
csr-969lt   21s   system:bootstrap:ibpecg   Approved,Issued
csr-9vsfw   69m   system:node:nodea         Approved,Issued
csr-l8sck   19s   system:node:nodeb         Pending

ubuntu@nodea:~$
```

Here we see two new CSRs (judging by their age). One corresponds to the `system:bootstrap` group, using the prefix from the bootstrap token. The other corresponds to a new node, nodeb, and is in the Pending state. The Pending CSR is a result of the

`--rotate-server-certificates` we passed to nodeb's kubelet. This is the request for the serving certificate.

The CSR approving controllers implemented in core Kubernetes do not approve node serving certificates for security reasons, so any kubelets using the `--rotate-server-certificates` argument need to run a custom approving controller, or you the user must manually approve the serving certificate requests.

Approve the CSR using `kubectl certificate approve` on the Pending CSR:

```
ubuntu@nodea:~$ kubectl certificate approve csr-l8sck

certificatesigningrequest.certificates.k8s.io/csr-l8sck approved

ubuntu@nodea:~$
```

Now check the node status:

```
ubuntu@nodea:~$ kubectl get nodes

NAME    STATUS     ROLES    AGE   VERSION
nodea   Ready      master   70m   v1.17.2
nodeb   Ready      <none>   36s   v1.17.2

ubuntu@nodea:~$
```

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
7cf5382f84b0        174e0e8ef23d           "/home/weave/launch.…"   30 seconds ago       Up 29
seconds                           k8s_weave_weave-net-zh2hb_kube-system_59d072f1-9616-4dcc-9d2e-
2c3790d1ca6f_1
6a27606a9863        weaveworks/weave-npc   "/usr/bin/launch.sh"     54 seconds ago       Up 53
seconds                           k8s_weave-npc_weave-net-zh2hb_kube-system_59d072f1-9616-4dcc-
9d2e-2c3790d1ca6f_0
f6480ad875f7        k8s.gcr.io/kube-proxy  "/usr/local/bin/kube…"   57 seconds ago       Up 56
seconds                           k8s_kube-proxy_kube-proxy-bs7jt_kube-system_31fe5131-d889-
40e6-b39f-c7838bb9e617_0
85428f0221ea        k8s.gcr.io/pause:3.1   "/pause"                 About a minute ago   Up
About a minute                       k8s_POD_kube-proxy-bs7jt_kube-system_31fe5131-d889-40e6-
b39f-c7838bb9e617_0
3761037bc924        k8s.gcr.io/pause:3.1   "/pause"                 About a minute ago   Up
About a minute                       k8s_POD_weave-net-zh2hb_kube-system_59d072f1-9616-4dcc-
9d2e-2c3790d1ca6f_0

ubuntu@nodeb:~$
```

9. Test the newly bootstrapped node

Now that our node has joined the cluster, let's test if the Kubelet can run pods on its own through its `staticPodPath`, /etc/kubernetes/manifests. This directory is declared in the kubelet's config.yaml under `/var/lib/kubelet`, and is only used if it is present in the config.yaml.

On *nodea*, generate a simple pod manifest without running the pod (`--dry-run`) in yaml (`-o yaml`):

```
ubuntu@nodea:~$ 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
    imagePullPolicy: IfNotPresent
    name: mypod
    ports:
    - containerPort: 80
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}

ubuntu@nodea:~$
```

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

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

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

apiVersion: v1
kind: Pod
metadata:
  labels:
    run: mypod
  name: mypod
spec:
  containers:
  - image: nginx
    name: mypod
    ports:
    - containerPort: 80
  dnsPolicy: ClusterFirst
  restartPolicy: Always

ubuntu@nodeb:~$
```

In your nodea terminal session, list the pods on the master:

```
ubuntu@nodea:~$ kubectl get pods

NAME          READY   STATUS              RESTARTS   AGE
mypod-nodeb   0/1     ContainerCreating   0          6s

ubuntu@nodea:~$ kubectl get pods

NAME          READY   STATUS    RESTARTS   AGE
mypod-nodeb   1/1     Running   0          7s

ubuntu@nodea:~$
```

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:~$
```

Now check the pod listing on nodea:

```
ubuntu@nodea:~$ kubectl get pods -o wide

No resources found in default namespace.

ubuntu@nodea:~$
```

We know that our kubelet is properly configured to run pods without the API server. In kubeadm clusters like ours, the manifest folder is /etc/kubernetes/manifests, but the kubelet configuration argument `staticPodPath:` (either passed as a flag to the kubelet or as part of the kubelet config.yaml) can be adjusted to change the static pod path.

10. Test the expanded cluster

On the master, list the pods in all namespaces:

```
ubuntu@nodea:~$ kubectl get pods --all-namespaces -o wide

NAMESPACE     NAME                              READY   STATUS    RESTARTS   AGE    IP
NODE    NOMINATED NODE   READINESS GATES
kube-system   coredns-6955765f44-42vk2          1/1     Running   0          138m   10.32.0.2
nodea   <none>           <none>
kube-system   coredns-6955765f44-xznb4          1/1     Running   0          138m   10.32.0.3
nodea   <none>           <none>
kube-system   etcd-nodea                        1/1     Running   0          138m   172.31.47.36
nodea   <none>           <none>
kube-system   kube-apiserver-nodea              1/1     Running   0          138m   172.31.47.36
nodea   <none>           <none>
kube-system   kube-controller-manager-nodea     1/1     Running   0          138m   172.31.47.36
nodea   <none>           <none>
kube-system   kube-proxy-blf96                  1/1     Running   0          138m   172.31.47.36
nodea   <none>           <none>
kube-system   kube-proxy-bs7jt                  1/1     Running   0          69m    172.31.42.149
nodeb   <none>           <none>
kube-system   kube-scheduler-nodea              1/1     Running   0          138m   172.31.47.36
nodea   <none>           <none>
kube-system   weave-net-j5qkj                   2/2     Running   0          138m   172.31.47.36
nodea   <none>           <none>
kube-system   weave-net-zh2hb                   2/2     Running   1          69m    172.31.42.149
nodeb   <none>           <none>

ubuntu@nodea:~$
```

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@nodea:~$ kubectl taint nodes $(hostname) node-role.kubernetes.io/master=iso:NoSchedule

nodea tainted

ubuntu@nodea:~$
```

> 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 nodea 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@nodea:~$ kubectl create deploy my-nginx --image=nginx:1.11
```

```
deployment.apps/my-nginx created

ubuntu@nodea:~$
```

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 should go to nodeb. Let's list the pod distribution again:

```
ubuntu@nodea:~$ kubectl get pods -o wide

NAME                        READY   STATUS    RESTARTS   AGE   IP          NODE    NOMINATED
NODE   READINESS GATES
my-nginx-5b7bb8ff4f-mh7gk   1/1     Running   0          3s    10.44.0.1   nodeb   <none>
<none>

ubuntu@nodea:~$
```

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@nodea:~$ kubectl taint nodes $(hostname) node-role.kubernetes.io/master-

node/nodea untainted

ubuntu@nodea:~$
```

Now scale up:

```
ubuntu@nodea:~$ kubectl scale deploy/my-nginx --replicas=2

deployment.extensions/my-nginx scaled

ubuntu@nodea:~$
```

Now let's check our pod distribution.

```
ubuntu@nodea:~$ kubectl get pods -o wide

NAME                        READY   STATUS              RESTARTS   AGE   IP          NODE
NOMINATED NODE   READINESS GATES
my-nginx-5b7bb8ff4f-8j6r5   0/1     ContainerCreating   0          7s    <none>      nodea
<none>           <none>
my-nginx-5b7bb8ff4f-mh7gk   1/1     Running             0          27s   10.44.0.1   nodeb
<none>           <none>

ubuntu@nodea:~$ kubectl get pods -o wide

NAME                        READY   STATUS    RESTARTS   AGE   IP          NODE    NOMINATED
NODE   READINESS GATES
my-nginx-5b7bb8ff4f-8j6r5   1/1     Running   0          9s    10.32.0.4   nodea   <none>
<none>
my-nginx-5b7bb8ff4f-mh7gk   1/1     Running   0          29s   10.44.0.1   nodeb   <none>
<none>

ubuntu@nodea:~$
```

With the master taint removed pods again are scheduled across all nodes. You have successfully expanded your kubeadm cluster manually using TLS bootstrapping!

Congratulations, you have completed the lab!