RX-M Cloud Native Consulting

Advanced Kubernetes

Lab 1 – Installing a Kubernetes cluster by hand

Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts. Kubernetes seeks to foster an ecosystem of components and tools that relieve the burden of running applications in public and private clouds. Kubernetes can run on a range of platforms, from your laptop, to VMs on a cloud provider, to racks of bare metal servers.

The effort required to set up a cluster varies from running a single command to installing and configuring individual programs on each node in your cluster. In this lab we will setup a single node Kubernetes cluster from source. Our installation has several prerequisites:

- **Linux** – Our lab system VM is preinstalled with Ubuntu 16.04, though most Linux distributions supporting modern container managers will work fine.
- **Docker** – Kubernetes will work with a variety of container managers but Docker is the most tested and widely deployed (Docker >=1.3 is required, but the latest Docker version is recommended).
- **etcd** – Kubernetes requires a distributed key/value store to manage discovery and cluster metadata; though Kubernetes was originally designed to make this function pluggable, etcd is the only practical option.
- **Go** – Kubernetes is a Go application. If you use prebuilt binaries there is no need to install Go but we will be using the source distribution of Kubernetes to gain access to various installation scripts and tools not provided with the released binaries. To build Kubernetes we will need to install Go version 1.3 or higher.
- **GCC** – When building from source we will also need to install the Gnu Compiler Collection which Go depends on.
- **Curl** – While wget, the Ubuntu preinstalled web downloader, is comparable to curl, many of the scripts provided with Kubernetes use curl so we will also need to install this utility.

We will begin by installing the API Service and the Kubelet on one machine. We will complete the lab by testing our cluster with a simple Pod. In later labs we'll add more nodes and Kubernetes services.

Lab 1A – Preparing the system components

In this first part we will prepare the system to run Kubernetes and build the Kubernetes binaries.

1. Setup the master node

We will use one VM in our cluster initially, we will call the node *nodea*.

We need to rename each of the VMs so that they are uniquely identified. To begin, set the host name for the master VM to *nodea*:

```
laptop$ chmod 400 k8s-adv-student.pem

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

...

ubuntu@nodea:~$ sudo hostnamectl set-hostname nodea

ubuntu@nodea:~$ hostname

nodea

ubuntu@nodea:~$ cat /etc/hostname

nodea
ubuntu@nodea:~$
```

Next we will need to add the host name to our /etc/hosts file so that we can use the name in networking operations. Look up your lab system's IP address:

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

2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc mq state UP group default qlen 1000
    link/ether 02:ef:63:d5:3b:be brd ff:ff:ff:ff:ff:ff
    inet 172.31.28.198/20 brd 172.31.31.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::ef:63ff:fed5:3bbe/64 scope link
       valid_lft forever preferred_lft forever
ubuntu@nodea:~$
```

Add your new hostname and external IP to `/etc/hosts`:

```
ubuntu@nodea:~$ sudo vim /etc/hosts
ubuntu@nodea:~$ cat /etc/hosts
```

```
127.0.0.1 localhost
172.31.28.198 nodea

# 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:

```
ubuntu@nodea:~$ ping -c 1 yahoo.com

PING yahoo.com (98.138.253.109) 56(84) bytes of data.
64 bytes from ir1.fp.vip.ne1.yahoo.com (98.138.253.109): icmp_seq=1 ttl=128 time=57.5 ms

--- yahoo.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 57.583/57.583/57.583/0.000 ms
ubuntu@nodea:~$
```

If you can not resolve public DNS names or reach the internet, debug your connectivity before continuing.

2. Install Docker

Docker supplies installation instructions for many platforms. The Ubuntu 16.04.1 installation guide can be found online here: https://docs.docker.com/install/linux/docker-ce/ubuntu/

As described in the installation guide, Docker requires a 64-bit system with a Linux kernel having version 3.10 or newer. Use the `uname` command to check the version of your kernel:

```
ubuntu@nodea:~$ uname -a

Linux nodea 4.4.0-1052-aws #61-Ubuntu SMP Mon Feb 12 23:05:58 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
```

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

To get the latest version of Docker we will use the Docker supplied packages. To setup the docker package repo we will need to install some utilities. Docker comes in two flavors these days, CE and EE. We will setup the free Community Edition (CE). The Enterprise Edition (EE) is the same at the core but requires a commercial license and offers support.

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

...

ubuntu@nodea:~$ sudo apt-get -y install apt-transport-https ca-certificates curl

...

ubuntu@nodea:~$
```

Now add the repo key so that apt can verify Docker packages:

```
ubuntu@nodea:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

OK
ubuntu@nodea:~$
```

Next we need to add the Docker repo to the apt package manager. We need to use the repo for our specific flavor and version of Linux. The lsb_release -cs sub-command prints the name of your Ubuntu version, like xenial or trusty. Add the Docker repo:

```
ubuntu@nodea:~$ sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

ubuntu@nodea:~$
```

Now update the package indexes to include the Docker packages made available by the repo:

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

```
...
Get:4 https://download.docker.com/linux/ubuntu xenial InRelease [66.2 kB]
Get:5 https://download.docker.com/linux/ubuntu xenial/stable amd64 Packages [7,361 B]

ubuntu@nodea:~$
```

Notice the new `https://download.docker.com/linux/ubuntu xenial/stable` repository above.

Finally install the latest version of Docker CE:

```
ubuntu@nodea:~$ sudo apt-get -y install docker-ce

...

ubuntu@nodea:~$
```

The `kubelet` runs as *root* and therefore has direct access to the Docker daemon socket interface. Normal user accounts must use the `sudo` command to run command line tools like "docker" as *root*. For our in-class purposes, eliminating the need for `sudo` execution of the docker command will simplify our practice sessions. To make it possible to connect to the local Docker daemon domain socket without sudo we need to add our user id to the *docker* group. To add the user named *ubuntu* to the *docker* group execute the following command:

```
ubuntu@nodea:~$ sudo usermod -aG docker ubuntu

ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ id ubuntu

uid=1000(ubuntu) gid=1000(ubuntu)
groups=1000(ubuntu),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),109(n
etdev),110(lxd),999(docker)
ubuntu@nodea:~$
```

As you can see from the `id ubuntu` command, the account *ubuntu* is now a member of the *docker* group. Now try running “id” without an account name:

```
ubuntu@nodea:~$ id

uid=1000(ubuntu) gid=1000(ubuntu)
groups=1000(ubuntu),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),109(n
etdev),110(lxd)
ubuntu@nodea:~$
```

While the *docker* group was added to your group list, your login shell maintains the old groups. After updating your *ubuntu* groups you will need to restart your login shell to ensure the changes take effect.

```
ubuntu@nodea:~$ exit

laptop$
```

3. Verify Docker operation

Check your Docker client version with the docker client *--version* switch:

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

...

ubuntu@nodea:~$ docker --version

Docker version 18.09.3, build 774a1f4
ubuntu@nodea:~$
```

Verify that the Docker server (daemon) is running using the system service interface. On Ubuntu 16 with systemd:

```
ubuntu@nodea:~$ systemctl status --all --full docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2019-03-26 02:48:04 UTC; 2min 25s ago
     Docs: https://docs.docker.com
```

```
 Main PID: 4178 (dockerd)
   CGroup: /system.slice/docker.service
           └─4178 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Mar 26 02:48:04 nodea dockerd[4178]: time="2019-03-26T02:48:04.430926292Z" level=warning msg="Your kernel does not
support swap memory limit"
Mar 26 02:48:04 nodea dockerd[4178]: time="2019-03-26T02:48:04.430992354Z" level=warning msg="Your kernel does not
support cgroup rt period"
Mar 26 02:48:04 nodea dockerd[4178]: time="2019-03-26T02:48:04.431003212Z" level=warning msg="Your kernel does not
support cgroup rt runtime"
Mar 26 02:48:04 nodea dockerd[4178]: time="2019-03-26T02:48:04.431381779Z" level=info msg="Loading containers:
start."
Mar 26 02:48:04 nodea dockerd[4178]: time="2019-03-26T02:48:04.545251413Z" level=info msg="Default bridge
(docker0) is assigned with an IP address 172.17.0.0/16. Daemon option --bip can be
Mar 26 02:48:04 nodea dockerd[4178]: time="2019-03-26T02:48:04.584017162Z" level=info msg="Loading containers:
done."
Mar 26 02:48:04 nodea dockerd[4178]: time="2019-03-26T02:48:04.616328655Z" level=info msg="Docker daemon"
commit=774a1f4 graphdriver(s)=overlay2 version=18.09.3
Mar 26 02:48:04 nodea dockerd[4178]: time="2019-03-26T02:48:04.616423978Z" level=info msg="Daemon has completed
initialization"
Mar 26 02:48:04 nodea dockerd[4178]: time="2019-03-26T02:48:04.640404229Z" level=info msg="API listen on
/var/run/docker.sock"
Mar 26 02:48:04 nodea systemd[1]: Started Docker Application Container Engine.

q

ubuntu@nodea:~$
```

Press 'q' to quit the log listing.

By default, the Docker daemon listens on a Unix domain socket for commands. This Unix domain socket is only accessible locally by *root* and the *docker* group by default. In our lab system the Docker command line client will communicate with the Docker daemon using this domain socket, requiring the user to be root or to be a member of the docker group. The `kubelet` will communicate with the Docker daemon over this domain socket as well, using the user root. Examine the permissions on the `docker.sock` socket file and the groups associated with your login ID.

```
ubuntu@nodea:~$ ls -l /var/run/docker.sock

srw-rw---- 1 root docker 0 Mar 26 18:43 /var/run/docker.sock
ubuntu@nodea:~$
```

By keeping the Docker daemon restricted to listening on the `docker.sock` socket (and not over network interfaces) we do not need to worry about securing Docker on the network, only securing the host. Now check the version of all parts of the Docker platform with the docker version command:

```
ubuntu@nodea:~$ docker version

Client:
 Version:           18.09.3
 API version:       1.39
 Go version:        go1.10.8
 Git commit:        774a1f4
 Built:             Thu Feb 28 06:40:58 2019
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          18.09.3
  API version:      1.39 (minimum version 1.12)
  Go version:       go1.10.8
  Git commit:       774a1f4
  Built:            Thu Feb 28 05:59:55 2019
  OS/Arch:          linux/amd64
  Experimental:     false
ubuntu@nodea:~$
```

The client version information is listed first followed by the server version information. Newer versions of Docker can support older versions of the Docker API. Docker version 17.12 uses API version 1.35 by default. The server in the above example reports support for Docker API versions back to 1.12. Docker v1.13 was released in Winter 2017 and was the last of the old Docker versioning releases. New Docker versions use the year and month as their version. For example, 17.03 is the March release in 2017. The API version have always been different from the Docker Engine versions and continue to monotonically increase (Docker 1.13 uses API 1.25, Docker 17.03.0 uses API 1.26 and Docker 17.03.1 uses API 1.27).

You can also use the Docker client to retrieve basic platform information from the Docker daemon:

```
ubuntu@nodea:~$ docker system info

Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
```

```
Images: 0
Server Version: 18.09.3
Storage Driver: overlay2
 Backing Filesystem: extfs
 Supports d_type: true
 Native Overlay Diff: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
 Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: e6b3f5632f50dbc4e9cb6288d911bf4f5e95b18e
runc version: 6635b4f0c6af3810594d2770f662f34ddc15b40d
init version: fec3683
Security Options:
 apparmor
 seccomp
  Profile: default
Kernel Version: 4.4.0-1075-aws
Operating System: Ubuntu 16.04.5 LTS
OSType: linux
Architecture: x86_64
CPUs: 2
Total Memory: 7.795GiB
Name: nodea
ID: TQ5X:T6PX:K2WX:LMX6:W44V:RPMB:DME6:AQJP:AMOD:45JW:HCD2:RCXA
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https://index.docker.io/v1/
Labels:
Experimental: false
Insecure Registries:
 127.0.0.0/8
Live Restore Enabled: false
Product License: Community Engine

WARNING: No swap limit support
```

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

- What version of Docker is your `docker` command line client?
- What version of Docker is your Docker Engine?
- What is the Logging Driver in use by your Docker Engine?
- What is the Storage Driver in use by your Docker Engine?
- What is the Root Directory used by your Storage Driver?
- What is the Runtime in use by your Docker Engine?
- Does the word “Driver” make you think that these component can be substituted?

4. Install etcd

Etcd is a distributed, consistent key-value store for shared configuration and service discovery. Etcd is:

- Simple: offering a curl'able API (gRPC)
- Secure: supporting optional SSL client cert authentication
- Fast: benchmarked at 1000s of writes/s per instance
- Reliable: uses the Raft distributed consensus algorithm

To install etcd we begin by downloading the etcd tarball for the platform and version tested with our K8s version: 64-bit Linux and v3.3.10 respectively:

```
ubuntu@nodea:~$ wget https://github.com/coreos/etcd/releases/download/v3.3.10/etcd-v3.3.10-linux-amd64.tar.gz

...

ubuntu@nodea:~$
```

Next extract the files:

```
ubuntu@nodea:~$ tar xzf etcd-v3.3.10-linux-amd64.tar.gz

ubuntu@nodea:~$
```

Finally copy the `etcd` daemon and the `etcdctl` client onto the path:

```
ubuntu@nodea:~$ sudo cp etcd-v3.3.10-linux-amd64/etcd* /usr/bin

ubuntu@nodea:~$
```

Now the Kubernetes startup script will be able to find and run `etcd`. We can now also run the `etcdctl` program, which is the command line client for `etcd`, giving us the ability to examine and modify the Kubernetes state (for testing and experimentation).

5. Install Golang

Go, also commonly referred to as golang, is an open source programming language developed at Google in 2007 by Robert Griesemer, Rob Pike, and Ken Thompson. Designed primarily for systems programming, it is a compiled, statically typed language in the tradition of C and C++, with garbage collection, various safety aspects and CSP-style concurrent programming features added. Almost all of the tools and services in the Docker portfolio are coded in Go, as is Kubernetes.

We will install Go to facilitate our Kubernetes installation. The Go version we will use depends on GCC, which is already installed on most modern linux distros. The build depends on make.

Check your gcc version:

```
ubuntu@nodea:~$ gcc --version

The program 'gcc' is currently not installed. You can install it by typing:
sudo apt install gcc
ubuntu@nodea:~$
```

You can generally install gcc and make from packages (`sudo apt-get install gcc`):

```
ubuntu@nodea:~$ sudo apt-get -y install gcc make

...
```

Next download Go binaries, we will use Go v1.12.1 which is the default Go version with our version of Kubernetes.

```
ubuntu@nodea:~$ wget https://storage.googleapis.com/golang/go1.12.1.linux-amd64.tar.gz

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

Extract the Go binaries into `/usr/local`:

```
ubuntu@nodea:~$ sudo tar -C /usr/local/ -xzf go1.12.1.linux-amd64.tar.gz

ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ ls -l /usr/local/go/

total 208
drwxr-xr-x  2 root root  4096 Mar 14 19:43 api
-rw-r--r--  1 root root 55358 Mar 14 19:43 AUTHORS
drwxr-xr-x  2 root root  4096 Mar 14 19:46 bin
-rw-r--r--  1 root root  1339 Mar 14 19:43 CONTRIBUTING.md
-rw-r--r--  1 root root 78132 Mar 14 19:43 CONTRIBUTORS
drwxr-xr-x  8 root root  4096 Mar 14 19:43 doc
-rw-r--r--  1 root root  5686 Mar 14 19:43 favicon.ico
drwxr-xr-x  3 root root  4096 Mar 14 19:43 lib
-rw-r--r--  1 root root  1479 Mar 14 19:43 LICENSE
drwxr-xr-x 13 root root  4096 Mar 14 19:43 misc
-rw-r--r--  1 root root  1303 Mar 14 19:43 PATENTS
drwxr-xr-x  6 root root  4096 Mar 14 19:46 pkg
-rw-r--r--  1 root root  1607 Mar 14 19:43 README.md
-rw-r--r--  1 root root    26 Mar 14 19:43 robots.txt
drwxr-xr-x 46 root root  4096 Mar 14 19:43 src
drwxr-xr-x 21 root root 12288 Mar 14 19:43 test
-rw-r--r--  1 root root     8 Mar 14 19:43 VERSION
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ ls -l /usr/local/go/bin/

total 34716
```

```
-rwxr-xr-x 1 root root 14609408 Mar 14 19:45 go
-rwxr-xr-x 1 root root 17409952 Mar 14 19:46 godoc
-rwxr-xr-x 1 root root  3525817 Mar 14 19:45 gofmt
ubuntu@nodea:~$
```

To wrap up the perquisites we need to add the go binaries to the system path. Add an export statement to the end of the system `/etc/profile` which appends `/usr/local/go/bin` to the *PATH* environment variable:

```
ubuntu@nodea:~$ sudo vim /etc/profile

ubuntu@nodea:~$ tail /etc/profile

if [ -d /etc/profile.d ]; then
  for i in /etc/profile.d/*.sh; do
    if [ -r $i ]; then
      . $i
    fi
  done
  unset i
fi

export PATH=$PATH:/usr/local/go/bin

ubuntu@nodea:~$
```

Apply the changes by either rebooting or source the change.

```
ubuntu@nodea:~$ source /etc/profile

ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ which go

/usr/local/go/bin/go
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ go version

go version go1.12.1 linux/amd64
ubuntu@nodea:~$
```

6. Build the Kubernetes binaries

With all of our prerequisites installed we can now turn our attention to installing Kubernetes. Kubernetes is packaged in several ways. You can download Kubernetes container images, install from packages, use tools (Ansible, etc.,) build from source or download prebuilt binaries to install by hand. In this example we'll download the source, build it and install the binaries ourselves.

To begin, download the source for Kubernetes:

```
ubuntu@nodea:~$ wget https://github.com/kubernetes/kubernetes/archive/v1.14.0.tar.gz

...

ubuntu@nodea:~$
```

Next extract the archive to the k8s directory:

```
ubuntu@nodea:~$ mkdir k8s

ubuntu@nodea:~$ tar xzf v1.14.0.tar.gz -C k8s/ --strip-components=1

ubuntu@nodea:~$
```

This creates a Kubernetes directory with all of the application sources, tests, build tooling and utilities.

```
ubuntu@nodea:~$ ls -l k8s/

total 240
drwxrwxr-x  4 ubuntu ubuntu   4096 Mar 21 05:51 api
drwxrwxr-x 13 ubuntu ubuntu   4096 Mar 21 05:51 build
```

```
lrwxrwxrwx  1 ubuntu ubuntu     21 Mar 21 05:51 BUILD.bazel -> build/root/BUILD.root
-rw-rw-r--  1 ubuntu ubuntu 131081 Mar 21 05:51 CHANGELOG-1.14.md
-rw-rw-r--  1 ubuntu ubuntu   1276 Mar 21 05:51 CHANGELOG.md
drwxrwxr-x 13 ubuntu ubuntu   4096 Mar 21 05:51 cluster
drwxrwxr-x 22 ubuntu ubuntu   4096 Mar 21 05:51 cmd
-rw-rw-r--  1 ubuntu ubuntu    148 Mar 21 05:51 code-of-conduct.md
-rw-rw-r--  1 ubuntu ubuntu    493 Mar 21 05:51 CONTRIBUTING.md
drwxrwxr-x  2 ubuntu ubuntu   4096 Mar 21 05:51 docs
drwxrwxr-x  2 ubuntu ubuntu   4096 Mar 21 05:51 Godeps
drwxrwxr-x 10 ubuntu ubuntu   4096 Mar 21 05:51 hack
-rw-rw-r--  1 ubuntu ubuntu  11358 Mar 21 05:51 LICENSE
drwxrwxr-x  2 ubuntu ubuntu   4096 Mar 21 05:51 logo
lrwxrwxrwx  1 ubuntu ubuntu     19 Mar 21 05:51 Makefile -> build/root/Makefile
lrwxrwxrwx  1 ubuntu ubuntu     35 Mar 21 05:51 Makefile.generated_files -> build/root/Makefile.generated_files
-rw-rw-r--  1 ubuntu ubuntu    645 Mar 21 05:51 OWNERS
-rw-rw-r--  1 ubuntu ubuntu   6149 Mar 21 05:51 OWNERS_ALIASES
drwxrwxr-x 34 ubuntu ubuntu   4096 Mar 21 05:51 pkg
drwxrwxr-x  3 ubuntu ubuntu   4096 Mar 21 05:51 plugin
-rw-rw-r--  1 ubuntu ubuntu   3179 Mar 21 05:51 README.md
-rw-rw-r--  1 ubuntu ubuntu    562 Mar 21 05:51 SECURITY_CONTACTS
drwxrwxr-x  4 ubuntu ubuntu   4096 Mar 21 05:51 staging
-rw-rw-r--  1 ubuntu ubuntu   1110 Mar 21 05:51 SUPPORT.md
drwxrwxr-x 15 ubuntu ubuntu   4096 Mar 21 05:51 test
drwxrwxr-x  7 ubuntu ubuntu   4096 Mar 21 05:51 third_party
drwxrwxr-x  4 ubuntu ubuntu   4096 Mar 21 05:51 translations
drwxrwxr-x 13 ubuntu ubuntu   4096 Mar 21 05:51 vendor
lrwxrwxrwx  1 ubuntu ubuntu     20 Mar 21 05:51 WORKSPACE -> build/root/WORKSPACE
ubuntu@nodea:~$
```

The hack directory under the Kubernetes tree root contains various scripts and tools designed to help developers work with Kubernetes. For example the `hack/local-up-cluster.sh` script is used to start a single node Kubernetes cluster.

We will build the minimal required components. The compile process consumes a sizable amount of memory. Our lab VM is configured with 8GB of RAM. This is enough to perform the compile. To build the binaries we will first change to a *root* shell and verify that the path includes Go.

```
ubuntu@nodea:~$ sudo su -

root@nodea:~#
```

```
root@nodea:~#  cd /home/ubuntu/k8s/

root@nodea:/home/ubuntu/k8s#
```

```
root@nodea:/home/ubuntu/k8s# which go

/usr/local/go/bin/go

root@nodea:/home/ubuntu/k8s#
```

Enter the make command to start the build (then take a break, the build will take up to 30 minutes).

```
root@nodea:/home/ubuntu/k8s# make

+++ [0326 02:56:26] Building go targets for linux/amd64:
    ./vendor/k8s.io/code-generator/cmd/deepcopy-gen
+++ [0326 02:56:33] Building go targets for linux/amd64:
    ./vendor/k8s.io/code-generator/cmd/defaulter-gen
+++ [0326 02:56:38] Building go targets for linux/amd64:
    ./vendor/k8s.io/code-generator/cmd/conversion-gen
+++ [0326 02:56:47] Building go targets for linux/amd64:
    ./vendor/k8s.io/kube-openapi/cmd/openapi-gen
+++ [0326 02:56:56] Building go targets for linux/amd64:
    ./vendor/github.com/jteeuwen/go-bindata/go-bindata
+++ [0326 02:56:57] Building go targets for linux/amd64:
    cmd/kube-proxy
    cmd/kube-apiserver
    cmd/kube-controller-manager
    cmd/cloud-controller-manager
    cmd/kubelet
    cmd/kubeadm
    cmd/hyperkube
    cmd/kube-scheduler
    vendor/k8s.io/apiextensions-apiserver
    cluster/gce/gci/mounter
    cmd/kubectl
    cmd/gendocs
    cmd/genkubedocs
    cmd/genman
    cmd/genyaml
    cmd/genswaggertypedocs
    cmd/linkcheck
    vendor/github.com/onsi/ginkgo/ginkgo
    test/e2e/e2e.test
    cmd/kubemark
    vendor/github.com/onsi/ginkgo/ginkgo
    test/e2e_node/e2e_node.test

root@nodea:/home/ubuntu/k8s#
```

The build will take up to 30 minutes. Go to lunch!

To build only a single binary use the *WHAT* flag, for example: `make all WHAT=cmd/kubemark`

When the build completes, return to the regular *ubuntu* account by entering `exit` in the root shell.

```
root@nodea:/home/ubuntu/k8s# exit

logout
ubuntu@nodea:~$
```

Lab 1B – Running the system

In this second part we will start up a Kubernetes cluster.

7. Run etcd

Kubernetes stores all its cluster state in etcd, a distributed data store with a strong consistency model. This state includes what nodes exist in the cluster, what pods should be running, which nodes they are running on, etc. The API server is the only Kubernetes component that connects to etcd; all the other components must go through the API server to work with cluster state. **Open up a new terminal** and run etcd:

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

...
```

```
ubuntu@nodea:~$ etcd

2019-03-26 03:05:53.915316 I | etcdmain: etcd Version: 3.3.10
2019-03-26 03:05:53.915367 I | etcdmain: Git SHA: 27fc7e2
2019-03-26 03:05:53.915372 I | etcdmain: Go Version: go1.10.4
2019-03-26 03:05:53.915388 I | etcdmain: Go OS/Arch: linux/amd64
2019-03-26 03:05:53.915397 I | etcdmain: setting maximum number of CPUs to 2, total number of available CPUs is 2
2019-03-26 03:05:53.915408 W | etcdmain: no data-dir provided, using default data-dir ./default.etcd
2019-03-26 03:05:53.915464 N | etcdmain: the server is already initialized as member before, starting as etcd
member...
2019-03-26 03:05:53.922425 I | embed: listening for peers on http://localhost:2380
2019-03-26 03:05:53.922487 I | embed: listening for client requests on localhost:2379
2019-03-26 03:05:53.924383 I | etcdserver: name = default
2019-03-26 03:05:53.924399 I | etcdserver: data dir = default.etcd
2019-03-26 03:05:53.924404 I | etcdserver: member dir = default.etcd/member
2019-03-26 03:05:53.924419 I | etcdserver: heartbeat = 100ms
2019-03-26 03:05:53.924426 I | etcdserver: election = 1000ms
2019-03-26 03:05:53.924433 I | etcdserver: snapshot count = 100000
2019-03-26 03:05:53.924449 I | etcdserver: advertise client URLs = http://localhost:2379
2019-03-26 03:05:53.924683 I | etcdserver: restarting member 8e9e05c52164694d in cluster cdf818194e3a8c32 at
commit index 4
2019-03-26 03:05:53.924712 I | raft: 8e9e05c52164694d became follower at term 2
2019-03-26 03:05:53.924727 I | raft: newRaft 8e9e05c52164694d [peers: [], term: 2, commit: 4, applied: 0,
lastindex: 4, lastterm: 2]
2019-03-26 03:05:53.930258 W | auth: simple token is not cryptographically signed
2019-03-26 03:05:53.931621 I | etcdserver: starting server... [version: 3.3.10, cluster version: to_be_decided]
2019-03-26 03:05:53.935807 I | etcdserver/membership: added member 8e9e05c52164694d [http://localhost:2380] to
cluster cdf818194e3a8c32
2019-03-26 03:05:53.935911 N | etcdserver/membership: set the initial cluster version to 3.3
2019-03-26 03:05:53.935947 I | etcdserver/api: enabled capabilities for version 3.3
2019-03-26 03:05:55.226042 I | raft: 8e9e05c52164694d is starting a new election at term 2
2019-03-26 03:05:55.226080 I | raft: 8e9e05c52164694d became candidate at term 3
2019-03-26 03:05:55.226102 I | raft: 8e9e05c52164694d received MsgVoteResp from 8e9e05c52164694d at term 3
2019-03-26 03:05:55.226120 I | raft: 8e9e05c52164694d became leader at term 3
2019-03-26 03:05:55.226132 I | raft: raft.node: 8e9e05c52164694d elected leader 8e9e05c52164694d at term 3
2019-03-26 03:05:55.227173 I | embed: ready to serve client requests
2019-03-26 03:05:55.228164 I | etcdserver: published {Name:default ClientURLs:[http://localhost:2379]} to cluster
cdf818194e3a8c32
2019-03-26 03:05:55.228184 E | etcdmain: forgot to set Type=notify in systemd service file?
2019-03-26 03:05:55.228974 N | embed: serving insecure client requests on 127.0.0.1:2379, this is strongly
discouraged!
...
```

Securing the control plane complicates the installation quite a bit. We have run etcd without TLS support for the time being. Later we will add security.

8. Run the Kubernetes master API server

To run the API server we will need to specify the location of the etcd server the API will use and the cluster IP address range for services to use.

Open a new tab or terminal and type the following to run the API Server:

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

...

ubuntu@nodea:~$ sudo $HOME/k8s/_output/bin/kube-apiserver \
--etcd-servers=http://localhost:2379 \
--allow-privileged=true \
--service-cluster-ip-range=10.0.0.0/16 \
--insecure-bind-address=0.0.0.0 \
--disable-admission-plugins=ServiceAccount

Flag --insecure-bind-address has been deprecated, This flag will be removed in a future version.
W0708 17:33:11.288837   38525 authentication.go:378] AnonymousAuth is not allowed with the AlwaysAllow authorizer.
Resetting AnonymousAuth to false. You should use a different authorizer
I0708 17:33:11.289368   38525 server.go:145] Version: v1.14.0
I0708 17:33:11.740021   38525 master.go:234] Using reconciler: lease
W0708 17:33:12.537462   38525 genericapiserver.go:319] Skipping API batch/v2alpha1 because it has no resources.
W0708 17:33:12.898374   38525 genericapiserver.go:319] Skipping API rbac.authorization.k8s.io/v1alpha1 because it
has no resources.
W0708 17:33:12.904626   38525 genericapiserver.go:319] Skipping API scheduling.k8s.io/v1alpha1 because it has no
resources.
W0708 17:33:12.951833   38525 genericapiserver.go:319] Skipping API storage.k8s.io/v1alpha1 because it has no
resources.
W0708 17:33:13.557893   38525 genericapiserver.go:319] Skipping API admissionregistration.k8s.io/v1alpha1 because
it has no resources.
[restful] 2018/07/08 17:33:13 log.go:33: [restful/swagger] listing is available at
https://192.168.225.229:6443/swaggerapi
[restful] 2018/07/08 17:33:13 log.go:33: [restful/swagger] https://192.168.225.229:6443/swaggerui/ is mapped to
folder /swagger-ui/
[restful] 2018/07/08 17:33:14 log.go:33: [restful/swagger] listing is available at
https://192.168.225.229:6443/swaggerapi
[restful] 2018/07/08 17:33:14 log.go:33: [restful/swagger] https://192.168.225.229:6443/swaggerui/ is mapped to
folder /swagger-ui/
```

```
I0708 17:33:17.250624   38525 insecure_handler.go:119] Serving insecurely on 0.0.0.0:8080
I0708 17:33:17.251749   38525 serve.go:96] Serving securely on [::]:6443
I0708 17:33:17.251860   38525 apiservice_controller.go:90] Starting APIServiceRegistrationController
I0708 17:33:17.251903   38525 cache.go:32] Waiting for caches to sync for APIServiceRegistrationController
controller
I0708 17:33:17.252942   38525 available_controller.go:278] Starting AvailableConditionController
I0708 17:33:17.253041   38525 cache.go:32] Waiting for caches to sync for AvailableConditionController controller
I0708 17:33:17.253967   38525 crd_finalizer.go:242] Starting CRDFinalizer
I0708 17:33:17.254270   38525 controller.go:84] Starting OpenAPI AggregationController
I0708 17:33:17.254341   38525 customresource_discovery_controller.go:199] Starting DiscoveryController
I0708 17:33:17.254357   38525 naming_controller.go:284] Starting NamingConditionController
I0708 17:33:17.254369   38525 establishing_controller.go:73] Starting EstablishingController
I0708 17:33:17.254371   38525 autoregister_controller.go:136] Starting autoregister controller
I0708 17:33:17.254413   38525 cache.go:32] Waiting for caches to sync for autoregister controller
I0708 17:33:17.259130   38525 crdregistration_controller.go:112] Starting crd-autoregister controller
I0708 17:33:17.259406   38525 controller_utils.go:1025] Waiting for caches to sync for crd-autoregister controller
W0708 17:33:17.301939   38525 lease.go:223] Resetting endpoints for master service "kubernetes" to
[192.168.225.229]
I0708 17:33:17.354734   38525 cache.go:39] Caches are synced for autoregister controller
I0708 17:33:17.355690   38525 cache.go:39] Caches are synced for APIServiceRegistrationController controller
I0708 17:33:17.355729   38525 cache.go:39] Caches are synced for AvailableConditionController controller
I0708 17:33:17.360802   38525 controller_utils.go:1032] Caches are synced for crd-autoregister controller
I0708 17:33:18.257148   38525 storage_scheduling.go:91] created PriorityClass system-node-critical with value
2000001000
I0708 17:33:18.259686   38525 storage_scheduling.go:91] created PriorityClass system-cluster-critical with value
2000000000
I0708 17:33:18.259701   38525 storage_scheduling.go:100] all system priority classes are created successfully or
already exist.
...
```

To test your new API server, open another terminal and use curl to retrieve the node list:

```
ubuntu@nodea:~$ curl http://localhost:8080/api/v1/nodes && echo

{
  "kind": "NodeList",
  "apiVersion": "v1",
  "metadata": {
    "selfLink": "/api/v1/nodes",
    "resourceVersion": "63"
  },
  "items": []
}
ubuntu@nodea:~$
```

As the curl response of `"items": null` indicates, we have no nodes. An API server with no nodes is not very interesting.

9. Create a node for the cluster

In Kubernetes a Node (previously known as a minion) is a compute host that Kubernetes can use to run containers. To make our API Server into a node we need to run the kubelet service. The kubelet acts as the node agent, running containers through calls to the Docker daemon.

Swap (vs memory limits)

As of 1.8 The kubelet fails if swap is enabled on a node. The release notes suggest:

> To override the default and run with /proc/swaps on, set failSwapOn: false

However, for our purposes we can simply turn off swap:

```
ubuntu@nodea:~$ sudo cat /proc/swaps

Filename                                Type            Size    Used    Priority
ubuntu@nodea:~$
```

Looks like we are already properly configured.

kubelet

Before starting the kubelet, we will create a configuration file using the kubeconfig syntax. You can learn more here: https://kubernetes.io/docs/concepts/configuration/organize-cluster-access-kubeconfig/.

Be sure to use your Kubernetes master hostname or VM IP in place of the hostname in the example below:

```
ubuntu@nodea:~$ vim nodea.conf
ubuntu@nodea:~$ cat nodea.conf

apiVersion: v1
```

```
clusters:
- cluster:
    server: http://nodea:8080
  name: local
contexts:
- context:
    cluster: local
    user: ""
  name: local
current-context: local
kind: Config
preferences: {}
users: []
ubuntu@nodea:~$
```

A subset of the Kubelet's configuration parameters may be set via an on-disk config file as a substitute for command-line flags. Providing parameters via a config file is the recommended approach because it simplifies node deployment and configuration management. You can learn more here: https://kubernetes.io/docs/tasks/administer-cluster/kubelet-config-file/

```
ubuntu@nodea:~$ vim nodea.yaml
ubuntu@nodea:~$ cat nodea.yaml

apiVersion: kubelet.config.k8s.io/v1beta1
kind: KubeletConfiguration
authentication:
  anonymous:
    enabled: true
cgroupDriver: cgroupfs
failSwapOn: true
ubuntu@nodea:~$
```

In a **new terminal** run the kubelet with following flags to use the kubeconfig:

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

...

ubuntu@nodea:~$ sudo $HOME/k8s/_output/bin/kubelet \
--kubeconfig=nodea.conf \
--config=nodea.yaml \
--allow-privileged=true \
--runtime-cgroups=/systemd/machine.slice \
--kubelet-cgroups=/systemd/machine.slice \
--pod-infra-container-image=k8s.gcr.io/pause:3.1

Flag --allow-privileged has been deprecated, will be removed in a future version
Flag --kubelet-cgroups has been deprecated, This parameter should be set via the config file specified by the
Kubelet's --config flag. See https://kubernetes.io/docs/tasks/administer-cluster/kubelet-config-file/ for more
information.
Flag --allow-privileged has been deprecated, will be removed in a future version
Flag --kubelet-cgroups has been deprecated, This parameter should be set via the config file specified by the
Kubelet's --config flag. See https://kubernetes.io/docs/tasks/administer-cluster/kubelet-config-file/ for more
information.
I0330 01:26:49.468564    4210 server.go:417] Version: v1.14.0
I0330 01:26:49.468851    4210 plugins.go:103] No cloud provider specified.
I0330 01:26:49.510329    4210 server.go:625] --cgroups-per-qos enabled, but --cgroup-root was not specified.
defaulting to /
I0330 01:26:49.510591    4210 container_manager_linux.go:261] container manager verified user specified cgroup-
root exists: []
I0330 01:26:49.510611    4210 container_manager_linux.go:266] Creating Container Manager object based on Node
Config: {RuntimeCgroupsName:/systemd/machine.slice SystemCgroupsName: KubeletCgroupsName:/systemd/machine.slice
ContainerRuntime:docker CgroupsPerQOS:true CgroupRoot:/ CgroupDriver:cgroupfs KubeletRootDir:/var/lib/kubelet
ProtectKernelDefaults:false NodeAllocatableConfig:{KubeReservedCgroupName: SystemReservedCgroupName:
EnforceNodeAllocatable:map[pods:{}] KubeReserved:map[] SystemReserved:map[] HardEvictionThresholds:
[{Signal:memory.available Operator:LessThan Value:{Quantity:100Mi Percentage:0} GracePeriod:0s MinReclaim:<nil>}
{Signal:nodefs.available Operator:LessThan Value:{Quantity:<nil> Percentage:0.1} GracePeriod:0s MinReclaim:<nil>}
{Signal:nodefs.inodesFree Operator:LessThan Value:{Quantity:<nil> Percentage:0.05} GracePeriod:0s MinReclaim:
<nil>} {Signal:imagefs.available Operator:LessThan Value:{Quantity:<nil> Percentage:0.15} GracePeriod:0s
MinReclaim:<nil>}]} QOSReserved:map[] ExperimentalCPUManagerPolicy:none ExperimentalCPUManagerReconcilePeriod:10s
ExperimentalPodPidsLimit:-1 EnforceCPULimits:true CPUCFSQuotaPeriod:100ms}
I0330 01:26:49.510703    4210 container_manager_linux.go:286] Creating device plugin manager: true
I0330 01:26:49.510722    4210 state_mem.go:36] [cpumanager] initializing new in-memory state store
I0330 01:26:49.510795    4210 state_mem.go:84] [cpumanager] updated default cpuset: ""
I0330 01:26:49.510811    4210 state_mem.go:92] [cpumanager] updated cpuset assignments: "map[]"
I0330 01:26:49.510918    4210 kubelet.go:304] Watching apiserver
I0330 01:26:49.514699    4210 client.go:75] Connecting to docker on unix:///var/run/docker.sock
I0330 01:26:49.514722    4210 client.go:104] Start docker client with request timeout=2m0s
W0330 01:26:49.515768    4210 docker_service.go:561] Hairpin mode set to "promiscuous-bridge" but kubenet is not
enabled, falling back to "hairpin-veth"
I0330 01:26:49.515792    4210 docker_service.go:238] Hairpin mode set to "hairpin-veth"
W0330 01:26:49.515904    4210 cni.go:213] Unable to update cni config: No networks found in /etc/cni/net.d
W0330 01:26:49.517338    4210 hostport_manager.go:68] The binary conntrack is not installed, this can cause
```

```
failures in network connection cleanup.
I0330 01:26:49.518316    4210 docker_service.go:253] Docker cri networking managed by kubernetes.io/no-op
I0330 01:26:49.534970    4210 docker_service.go:258] Docker Info: &
{ID:TQ5X:T6PX:K2WX:LMX6:W44V:RPMB:DME6:AQJP:AMOD:45JW:HCD2:RCXA Containers:0 ContainersRunning:0
ContainersPaused:0 ContainersStopped:0 Images:3 Driver:overlay2 DriverStatus:[[Backing Filesystem extfs] [Supports
d_type true] [Native Overlay Diff true]] SystemStatus:[] Plugins:{Volume:[local] Network:[bridge host macvlan null
overlay] Authorization:[] Log:[awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog]}
MemoryLimit:true SwapLimit:false KernelMemory:true CPUCfsPeriod:true CPUCfsQuota:true CPUShares:true CPUSet:true
IPv4Forwarding:true BridgeNfIptables:true BridgeNfIP6tables:true Debug:false NFd:22 OomKillDisable:true
NGoroutines:38 SystemTime:2019-03-30T01:26:49.518976364Z LoggingDriver:json-file CgroupDriver:cgroupfs
NEventsListener:0 KernelVersion:4.4.0-1075-aws OperatingSystem:Ubuntu 16.04.5 LTS OSType:linux Architecture:x86_64
IndexServerAddress:https://index.docker.io/v1/ RegistryConfig:0xc0002c6690 NCPU:2 MemTotal:8369913856
GenericResources:[] DockerRootDir:/var/lib/docker HTTPProxy: HTTPSProxy: NoProxy: Name:nodea Labels:[]
ExperimentalBuild:false ServerVersion:18.09.3 ClusterStore: ClusterAdvertise: Runtimes:map[runc:{Path:runc Args:
[]}] DefaultRuntime:runc Swarm:{NodeID: NodeAddr: LocalNodeState:inactive ControlAvailable:false Error:
RemoteManagers:[] Nodes:0 Managers:0 Cluster:<nil>} LiveRestoreEnabled:false Isolation: InitBinary:docker-init
ContainerdCommit:{ID:e6b3f5632f50dbc4e9cb6288d911bf4f5e95b18e Expected:e6b3f5632f50dbc4e9cb6288d911bf4f5e95b18e}
RuncCommit:{ID:6635b4f0c6af3810594d2770f662f34ddc15b40d Expected:6635b4f0c6af3810594d2770f662f34ddc15b40d}
InitCommit:{ID:fec3683 Expected:fec3683} SecurityOptions:[name=apparmor name=seccomp,profile=default]}
I0330 01:26:49.535100    4210 docker_service.go:271] Setting cgroupDriver to cgroupfs
I0330 01:26:49.553386    4210 remote_runtime.go:62] parsed scheme: ""
I0330 01:26:49.553408    4210 remote_runtime.go:62] scheme "" not registered, fallback to default scheme
I0330 01:26:49.553443    4210 remote_image.go:50] parsed scheme: ""
I0330 01:26:49.553454    4210 remote_image.go:50] scheme "" not registered, fallback to default scheme
I0330 01:26:49.553516    4210 asm_amd64.s:1337] ccResolverWrapper: sending new addresses to cc:
[{/var/run/dockershim.sock 0  <nil>}]
I0330 01:26:49.553564    4210 clientconn.go:796] ClientConn switching balancer to "pick_first"
I0330 01:26:49.553616    4210 balancer_conn_wrappers.go:131] pickfirstBalancer: HandleSubConnStateChange:
0xc0003542b0, CONNECTING
I0330 01:26:49.553672    4210 asm_amd64.s:1337] ccResolverWrapper: sending new addresses to cc:
[{/var/run/dockershim.sock 0  <nil>}]
I0330 01:26:49.553690    4210 clientconn.go:796] ClientConn switching balancer to "pick_first"
I0330 01:26:49.553723    4210 balancer_conn_wrappers.go:131] pickfirstBalancer: HandleSubConnStateChange:
0xc000261740, CONNECTING
I0330 01:26:49.553748    4210 balancer_conn_wrappers.go:131] pickfirstBalancer: HandleSubConnStateChange:
0xc0003542b0, READY
I0330 01:26:49.553866    4210 balancer_conn_wrappers.go:131] pickfirstBalancer: HandleSubConnStateChange:
0xc000261740, READY
I0330 01:26:49.554890    4210 kuberuntime_manager.go:210] Container runtime docker initialized, version: 18.09.3,
apiVersion: 1.39.0
I0330 01:26:49.556788    4210 server.go:1037] Started kubelet
E0330 01:26:49.556884    4210 kubelet.go:1282] Image garbage collection failed once. Stats initialization may not
have completed yet: failed to get imageFs info: unable to find data in memory cache
```

```
I0330 01:26:49.557260    4210 server.go:141] Starting to listen on 0.0.0.0:10250
I0330 01:26:49.557380    4210 fs_resource_analyzer.go:64] Starting FS ResourceAnalyzer
I0330 01:26:49.557454    4210 status_manager.go:152] Starting to sync pod status with apiserver
I0330 01:26:49.557501    4210 kubelet.go:1806] Starting kubelet main sync loop.
I0330 01:26:49.557551    4210 kubelet.go:1823] skipping pod synchronization - [container runtime status check may
not have completed yet., PLEG is not healthy: pleg has yet to be successful.]
I0330 01:26:49.557810    4210 volume_manager.go:248] Starting Kubelet Volume Manager
I0330 01:26:49.557897    4210 server.go:343] Adding debug handlers to kubelet server.
I0330 01:26:49.559002    4210 desired_state_of_world_populator.go:130] Desired state populator starts to run
I0330 01:26:49.583759    4210 clientconn.go:440] parsed scheme: "unix"
I0330 01:26:49.583778    4210 clientconn.go:440] scheme "unix" not registered, fallback to default scheme
I0330 01:26:49.583810    4210 asm_amd64.s:1337] ccResolverWrapper: sending new addresses to cc:
[{unix:///run/containerd/containerd.sock 0  <nil>}]
I0330 01:26:49.583846    4210 clientconn.go:796] ClientConn switching balancer to "pick_first"
I0330 01:26:49.583881    4210 balancer_conn_wrappers.go:131] pickfirstBalancer: HandleSubConnStateChange:
0xc0009596b0, CONNECTING
I0330 01:26:49.583991    4210 balancer_conn_wrappers.go:131] pickfirstBalancer: HandleSubConnStateChange:
0xc0009596b0, READY
I0330 01:26:49.657732    4210 kubelet.go:1823] skipping pod synchronization - container runtime status check may
not have completed yet.
I0330 01:26:49.658027    4210 kubelet_node_status.go:283] Setting node annotation to enable volume controller
attach/detach
I0330 01:26:49.658721    4210 cpu_manager.go:155] [cpumanager] starting with none policy
I0330 01:26:49.658738    4210 cpu_manager.go:156] [cpumanager] reconciling every 10s
I0330 01:26:49.658753    4210 policy_none.go:42] [cpumanager] none policy: Start
W0330 01:26:49.659429    4210 manager.go:538] Failed to retrieve checkpoint for "kubelet_internal_checkpoint":
checkpoint is not found
I0330 01:26:49.659807    4210 kubelet_node_status.go:72] Attempting to register node nodea
W0330 01:26:49.660089    4210 container_manager_linux.go:818] CPUAccounting not enabled for pid: 4210
W0330 01:26:49.660114    4210 container_manager_linux.go:821] MemoryAccounting not enabled for pid: 4210
I0330 01:26:49.683912    4210 kubelet_node_status.go:114] Node nodea was previously registered
I0330 01:26:49.683931    4210 kubelet_node_status.go:75] Successfully registered node nodea
I0330 01:26:49.870934    4210 reconciler.go:154] Reconciler: start to sync state
```

Read through the kubelet's startup log output. Note the following:

- The kubelet registers itself with the api-server: "`Successfully registered node nodea`"
- The kubelet automatically connects to docker on the Unix domain socket: "`Connecting to docker on unix:///var/run/docker.sock`"
- The kubelet is listening on port 10250: "`Starting to listen on 0.0.0.0:10250`"

Now repeat the node list request issued earlier in a new terminal:

```
ubuntu@nodea:~$ curl http://localhost:8080/api/v1/nodes && echo

{
  "kind": "NodeList",
  "apiVersion": "v1",
  "metadata": {
    "selfLink": "/api/v1/nodes",
    "resourceVersion": "74"
  },
  "items": [
    {
      "metadata": {
        "name": "nodea",
        "selfLink": "/api/v1/nodes/nodea",
        "uid": "23b8f43b-5250-11e9-b57b-02ef63d53bbe",
        "resourceVersion": "70",
        "creationTimestamp": "2019-03-29T18:26:16Z",
        "labels": {
          "beta.kubernetes.io/arch": "amd64",
          "beta.kubernetes.io/os": "linux",
          "kubernetes.io/arch": "amd64",
          "kubernetes.io/hostname": "nodea",
          "kubernetes.io/os": "linux"
        },
        "annotations": {
          "volumes.kubernetes.io/controller-managed-attach-detach": "true"
        }
      },
      "spec": {
        "taints": [
          {
            "key": "node.kubernetes.io/not-ready",
            "effect": "NoSchedule"
          }
        ]
      },
      "status": {
        "capacity": {
          "cpu": "2",
          "ephemeral-storage": "30428648Ki",
          "hugepages-2Mi": "0",
          "memory": "8173744Ki",
        "pods": "110"
      },
      "allocatable": {
        "cpu": "2",
        "ephemeral-storage": "28043041951",
        "hugepages-2Mi": "0",
        "memory": "8071344Ki",
        "pods": "110"
      },
      "conditions": [
        {
          "type": "MemoryPressure",
          "status": "False",
          "lastHeartbeatTime": "2019-03-29T18:27:16Z",
          "lastTransitionTime": "2019-03-29T18:26:16Z",
          "reason": "KubeletHasSufficientMemory",
          "message": "kubelet has sufficient memory available"
        },
        {
          "type": "DiskPressure",
          "status": "False",
          "lastHeartbeatTime": "2019-03-29T18:27:16Z",
          "lastTransitionTime": "2019-03-29T18:26:16Z",
          "reason": "KubeletHasNoDiskPressure",
          "message": "kubelet has no disk pressure"
        },
        {
          "type": "PIDPressure",
          "status": "False",
          "lastHeartbeatTime": "2019-03-29T18:27:16Z",
          "lastTransitionTime": "2019-03-29T18:26:16Z",
          "reason": "KubeletHasSufficientPID",
          "message": "kubelet has sufficient PID available"
        },
        {
          "type": "Ready",
          "status": "True",
          "lastHeartbeatTime": "2019-03-29T18:27:16Z",
          "lastTransitionTime": "2019-03-29T18:26:16Z",
          "reason": "KubeletReady",
          "message": "kubelet is posting ready status. AppArmor enabled"
        }
      ],
        "addresses": [
          {
            "type": "InternalIP",
            "address": "172.31.28.198"
          },
          {
            "type": "Hostname",
            "address": "nodea"
          }
        ],
        "daemonEndpoints": {
          "kubeletEndpoint": {
            "Port": 10250
          }
        },
        "nodeInfo": {
          "machineID": "e53d14d788454608be05a016cbffebf6",
          "systemUUID": "EC203559-73E9-971D-B8A9-50080CBED047",
          "bootID": "fe03b0b8-f5d2-490d-a949-faef5f3f1211",
          "kernelVersion": "4.4.0-1075-aws",
          "osImage": "Ubuntu 16.04.5 LTS",
          "containerRuntimeVersion": "docker://18.9.3",
          "kubeletVersion": "v1.14.0",
          "kubeProxyVersion": "v1.14.0",
          "operatingSystem": "linux",
          "architecture": "amd64"
        }
      }
    }
  ]
}
ubuntu@nodea:~$
```

- What conditions are being tracked?
- For each condition, what is the current value?
- What are the capacities of your node?

10. kubectl

Though curl is useful for examining the Kubernetes REST API directly, we can control the cluster more readily with the `kubectl` command line tool. Copy the kubectl command to the system path for easy access:

```
ubuntu@nodea:~$ sudo cp $HOME/k8s/_output/bin/kubectl /usr/bin/

ubuntu@nodea:~$
```

As before, display the nodes available on the local cluster, this time via `kubectl`:

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

NAME    STATUS   ROLES    AGE     VERSION
nodea   Ready    <none>   2m34s   v1.14.0
ubuntu@nodea:~$
```

Display detailed information about *nodea*.

```
ubuntu@nodea:~$ kubectl describe node nodea

Name:               nodea
Roles:              <none>
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/arch=amd64
                    kubernetes.io/hostname=nodea
                    kubernetes.io/os=linux
Annotations:        volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp:  Fri, 29 Mar 2019 18:26:16 +0000
Taints:             node.kubernetes.io/not-ready:NoSchedule
Unschedulable:      false
Conditions:
  Type             Status  LastHeartbeatTime                 LastTransitionTime                Reason
Message
  ----             ------  -----------------                 ------------------                ------
-------
  MemoryPressure   False   Fri, 29 Mar 2019 18:29:16 +0000   Fri, 29 Mar 2019 18:26:16 +0000
KubeletHasSufficientMemory   kubelet has sufficient memory available
  DiskPressure     False   Fri, 29 Mar 2019 18:29:16 +0000   Fri, 29 Mar 2019 18:26:16 +0000
KubeletHasNoDiskPressure     kubelet has no disk pressure
  PIDPressure      False   Fri, 29 Mar 2019 18:29:16 +0000   Fri, 29 Mar 2019 18:26:16 +0000
KubeletHasSufficientPID      kubelet has sufficient PID available
```

```
  Ready            True    Fri, 29 Mar 2019 18:29:16 +0000   Fri, 29 Mar 2019 18:26:16 +0000   KubeletReady
kubelet is posting ready status. AppArmor enabled
Addresses:
  InternalIP:  172.31.28.198
  Hostname:    nodea
Capacity:
 cpu:                2
 ephemeral-storage:  30428648Ki
 hugepages-2Mi:      0
 memory:             8173744Ki
 pods:               110
Allocatable:
 cpu:                2
 ephemeral-storage:  28043041951
 hugepages-2Mi:      0
 memory:             8071344Ki
 pods:               110
System Info:
 Machine ID:                 e53d14d788454608be05a016cbffebf6
 System UUID:                EC203559-73E9-971D-B8A9-50080CBED047
 Boot ID:                    fe03b0b8-f5d2-490d-a949-faef5f3f1211
 Kernel Version:             4.4.0-1075-aws
 OS Image:                   Ubuntu 16.04.5 LTS
 Operating System:           linux
 Architecture:               amd64
 Container Runtime Version:  docker://18.9.3
 Kubelet Version:            v1.14.0
 Kube-Proxy Version:         v1.14.0
Non-terminated Pods:         (0 in total)
  Namespace                  Name    CPU Requests  CPU Limits  Memory Requests  Memory Limits  AGE
  ---------                  ----    ------------  ----------  ---------------  -------------  ---
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource           Requests  Limits
  --------           --------  ------
  cpu                0 (0%)    0 (0%)
  memory             0 (0%)    0 (0%)
  ephemeral-storage  0 (0%)    0 (0%)
Events:
  Type    Reason                   Age                    From            Message
  ----    ------                   ----                   ----            -------
  Normal  Starting                 3m13s                  kubelet, nodea  Starting kubelet.
  Normal  NodeHasSufficientMemory  3m13s (x2 over 3m13s)  kubelet, nodea  Node nodea status is now:
```

```
NodeHasSufficientMemory
  Normal  NodeHasNoDiskPressure    3m13s (x2 over 3m13s)  kubelet, nodea  Node nodea status is now:
NodeHasNoDiskPressure
  Normal  NodeHasSufficientPID     3m13s (x2 over 3m13s)  kubelet, nodea  Node nodea status is now:
NodeHasSufficientPID
  Normal  NodeAllocatableEnforced  3m13s                  kubelet, nodea  Updated Node Allocatable limit across
pods
  Normal  NodeReady                3m13s                  kubelet, nodea  Node nodea status is now: NodeReady
ubuntu@nodea:~$
```

11. Run a pod on the cluster

We now have a working cluster!

To test our cluster further we should run a Pod. Create a simple Pod spec with your favorite editor.

```
ubuntu@nodea:~$ vim testpod.yaml
ubuntu@nodea:~$ cat testpod.yaml

apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  nodeName: nodea
  automountServiceAccountToken: false
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
    volumeMounts:
    - mountPath: /var/log/nginx
      name: nginx-logs
  - name: log-truncator
    image: busybox
    command:
    - /bin/sh
    args: [-c, 'while true; do cat /dev/null > /logdir/access.log; sleep 10; done']
    volumeMounts:
    - mountPath: /logdir
```

```
      name: nginx-logs
  volumes:
  - name: nginx-logs
    emptyDir: {}
ubuntu@nodea:~$
```

Now use `kubectl` to create your pod.

```
ubuntu@nodea:~$ kubectl create -f testpod.yaml

pod/nginx created
ubuntu@nodea:~$
```

Verify that your pod is running (if you see status "ContainerCreating", it is likely that docker is pulling your container image from Docker Hub, give it a minute or so to complete the download):

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

NAME      READY     STATUS              RESTARTS   AGE
nginx     0/2       ContainerCreating   0          15s
ubuntu@nodea:~$
```

Wait for it!!!

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

NAME      READY     STATUS    RESTARTS   AGE
nginx     2/2       Running   0          33s
ubuntu@nodea:~$
```

The pod is up! Wait, how is this possible?

- Is your cluster running the Kubernetes Scheduler?
- Does the pod you ran need scheduling? (Hint: `nodeName: nodea`)
- Is your cluster running the Kubernetes Controller Manager?

- Did you ask for a Deployment or a Replica Set?
- Is your cluster running kube-proxy?
- Are you using any services?

Kubernetes is a true microservice application, you can run some or all of the parts. As we will see in a later lab, you can run the `kubelet` by itself and use it for many test cases.

In this example, we have the `kubelet` and the API server running and those are the only parts we need, so everything works. Kubernetes is a cloud native application with a microservice architecture. Each service stands on its own and, while each service tries to connect to other services it knows how to use, microservices do not fail when they can't reach their counterparts, they just do what they can and continue to try to reach their counterparts in the background until they are available.

Get details about your pod:

```
ubuntu@nodea:~$ kubectl describe pod nginx

Name:               nginx
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:               nodea/172.31.28.198
Start Time:         Fri, 29 Mar 2019 18:51:50 +0000
Labels:             <none>
Annotations:        <none>
Status:             Running
IP:                 172.17.0.2
Containers:
  nginx:
    Container ID:   docker://70a5482f0ea353ca47402524dfc1a002b3e6361abf67f891eff3d349bd7bcc0d
    Image:          nginx
    Image ID:       docker-
pullable://nginx@sha256:c8a861b8a1eeef6d48955a6c6d5dff8e2580f13ff4d0f549e082e7c82a8617a2
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Fri, 29 Mar 2019 18:51:56 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/log/nginx from nginx-logs (rw)
  log-truncator:
    Container ID:  docker://5686be3ce8fa505acb0ef1cd164a53ea5eea78661f8afc01e6127ccae74534b2
    Image:         busybox
    Image ID:      docker-
pullable://busybox@sha256:061ca9704a714ee3e8b80523ec720c64f6209ad3f97c0ff7cb9ec7d19f15149f
    Port:          <none>
    Host Port:     <none>
    Command:
      /bin/sh
    Args:
      -c
      while true; do cat /dev/null > /logdir/access.log; sleep 10; done
    State:          Running
      Started:      Fri, 29 Mar 2019 18:51:58 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /logdir from nginx-logs (rw)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  nginx-logs:
    Type:       EmptyDir (a temporary directory that shares a pod's lifetime)
    Medium:
    SizeLimit:  <unset>
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason            Age                From           Message
  ----    ------            ----               ----           -------
  Normal  Pulling           61s                kubelet, nodea  Pulling image "nginx"
  Normal  Pulled            57s                kubelet, nodea  Successfully pulled image "nginx"
  Normal  Created           57s                kubelet, nodea  Created container nginx
  Normal  Started           56s                kubelet, nodea  Started container nginx
  Normal  Pulling           56s                kubelet, nodea  Pulling image "busybox"
  Normal  Pulled            54s                kubelet, nodea  Successfully pulled image "busybox"
  Normal   Created            54s                 kubelet, nodea  Created container log-truncator
  Normal   Started            54s                 kubelet, nodea  Started container log-truncator
  Warning  MissingClusterDNS  52s (x4 over 62s)  kubelet, nodea  pod: "nginx_default(b6057a26-5253-11e9-b57b-
02ef63d53bbe)". kubelet does not have ClusterDNS IP configured and cannot create Pod using "ClusterFirst" policy.
Falling back to "Default" policy.
ubuntu@nodea:~$
```

Now try to reach the web server with `curl`.

```
ubuntu@nodea:~$ kubectl get pod nginx -o custom-columns='IP:{.status.podIP}'

IP
172.17.0.2
ubuntu@nodea:~$
```

```
ubuntu@nodea:~$ curl -I 172.17.0.2

HTTP/1.1 200 OK
Server: nginx/1.13.8
Date: Wed, 17 Jan 2018 19:57:01 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 26 Dec 2018 11:11:22 GMT
Connection: keep-alive
ETag: "5a422e5a-264"
Accept-Ranges: bytes
ubuntu@nodea:~$
```

For the more adventurous, we can use the API to retrieve the IP. First let's install jq, a handy tool used to format and filter JSON strings.

```
ubuntu@nodea:~$ sudo apt-get install jq -y

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

Now curl the pods route and tell jq to display the hostIP and podIP fields found in the status element of the items array.

```
ubuntu@nodea:~$ curl -s http://localhost:8080/api/v1/pods \
| jq -r '[.items[].status | to_entries[] | select(.key | endswith("IP"))] | from_entries'

{
  "hostIP": "172.31.28.198",
  "podIP": "172.17.0.2"
}
ubuntu@nodea:~$
```

You can also use the kubectl command with the "custom-columns" output specifier:

```
ubuntu@nodea:~$ kubectl get pod \
-o=custom-columns=Name:.metadata.name,hostIP:.status.hostIP,podIP:status.podIP

Name      hostIP            podIP
nginx     172.31.28.198      172.17.0.2
ubuntu@nodea:~$
```

Or simply using the `-o wide` specifier.

Notice the Pod IP address is from the docker0 bridge, via `docker network` subcommand.

```
ubuntu@nodea:~$ docker network inspect bridge \
| jq -r '.[] | select(.Name=="bridge").IPAM.Config[].Subnet'

172.17.0.0/16
ubuntu@nodea:~$
```

or via the Docker API and jq:

```
ubuntu@nodea:~$ curl -s \
--unix-socket /var/run/docker.sock http:/networks/bridge \
| jq -r '.IPAM.Config[].Subnet'
```

```
172.17.0.0/16
ubuntu@nodea:~$
```

If you don't like all of the jq trickery you can of course just run the `docker network inspect bridge` command to display the docker networks.

Great! We have a minimal cluster with a running Pod. We'll add more parts to our cluster and learn more about Kubernetes in the labs ahead.

Congratulations you have completed the lab!