

# 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  $\geq 1.3$  is required, but the latest Docker version is recommended).
- **etcd** – Kubernetes requires a distributed key/value store to manage discovery and cluster metadata, we will use etcd for this purpose.
- **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*.

To run the lab virtual machine:

If the Lab VM image was not provided to you during class, instructions for downloading the lab VM can be found here: <https://github.com/RX-M/classfiles/blob/master/lab-setup.md>

Launch a copy of the Lab system VM to use as the Kubernetes cluster master. Login to the VM with the user name "user" and the password "user".

There is no reason to update this lab system and doing so may require large downloads. If the VM prompts you to perform a system update, choose the "Remind me later" option to avoid tying up the class room Internet connection.

The Launch Pad on the left side of the desktop has short cuts to commonly used programs. Click the Terminal icon to launch a new Bash command line shell.

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*:

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

user@ubuntu:~$ hostname

nodea

user@nodea:~$ cat /etc/hostname

nodea

user@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:

```
user@ubuntu:~$ ip a show ens33

2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:49:13:80 brd ff:ff:ff:ff:ff:ff
    inet 192.168.121.134/24 brd 192.168.121.255 scope global ens33
        valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe49:1380/64 scope link
```

```
valid_lft forever preferred_lft forever
```

```
user@ubuntu:~$
```

Add your new hostname and external IP to `/etc/hosts`, also comment out `127.0.1.1` line.

```
user@ubuntu:~$ sudo vim /etc/hosts
```

```
user@ubuntu:~$ cat /etc/hosts
```

```
127.0.0.1    localhost
```

```
#127.0.1.1   ubuntu
```

```
192.168.121.134 nodea
```

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

```
::1    localhost ip6-localhost ip6-loopback
```

```
ff02::1 ip6-allnodes
```

```
ff02::2 ip6-allrouters
```

```
user@ubuntu:~$
```

Finally, verify that you can reach the Internet:

```
user@ubuntu:~$ 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
```

```
user@ubuntu:~$
```

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/engine/installation/linux/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:

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

Linux ubuntu 4.4.0-93-generic #116-Ubuntu SMP Fri Aug 11 21:17:51 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux

user@ubuntu:~$
```

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.

If you receive the error: "E: Unable to lock the administration directory (/var/lib/dpkg/), is another process using it?", your system is probably performing apt initialization in the background. If you wait a minute or two for all of the processes running /usr/bin/dpkg to exit, you should then be able to perform the installation.

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

...

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

...

user@ubuntu:~$
```

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

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

OK
```

```
user@ubuntu:~$
```

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:

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

user@ubuntu:~$
```

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

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

...

user@ubuntu:~$
```

Finally install the latest version of Docker CE:

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

...

user@ubuntu:~$
```

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 `user` to the `docker` group execute the following command:

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

```
user@ubuntu:~$
```

```
user@ubuntu:~$ id user
```

```
uid=1000(user) gid=1000(user)  
groups=1000(user),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),110(lxd),115(lpadmin),116(sambashare),999(docker)
```

```
user@ubuntu:~$
```

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

```
user@ubuntu:~$ id
```

```
uid=1000(user) gid=1000(user)  
groups=1000(user),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),110(lxd),115(lpadmin),116(sambashare)
```

```
user@ubuntu:~$
```

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

```
user@ubuntu:~$ sudo reboot
```

When the system has restarted log back in as *user* with the password *user*.

### 3. Verify Docker operation

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

```
user@nodea:~$ docker --version
```

```
Docker version 17.09.0-ce, build afdb6d4
```

```
user@nodea:~$
```

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

```
user@nodea:~$ sudo systemctl status docker
```

● docker.service – Docker Application Container Engine

Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)

Active: active (running) since Fri 2017-10-06 09:37:46 PDT; 45s ago

Docs: <https://docs.docker.com>

Main PID: 1310 (dockerd)

Tasks: 18

Memory: 57.8M

CPU: 374ms

CGroup: /system.slice/docker.service

└─1310 /usr/bin/dockerd -H fd://

└─1417 docker-containerd -l unix:///var/run/docker/libcontainerd/docker-containerd.sock --metrics-interval=0 --start-timeout 2m --state-dir /var/run/docker/libcontainerd/contain

Oct 06 09:37:46 ubuntu dockerd[1310]: time="2017-10-06T09:37:46.439213597-07:00" level=warning msg="Your kernel does not support swap memory limit"

Oct 06 09:37:46 ubuntu dockerd[1310]: time="2017-10-06T09:37:46.439378704-07:00" level=warning msg="Your kernel does not support cgroup rt period"

Oct 06 09:37:46 ubuntu dockerd[1310]: time="2017-10-06T09:37:46.439488594-07:00" level=warning msg="Your kernel does not support cgroup rt runtime"

Oct 06 09:37:46 ubuntu dockerd[1310]: time="2017-10-06T09:37:46.439886657-07:00" level=info msg="Loading containers: start."

Oct 06 09:37:46 ubuntu dockerd[1310]: time="2017-10-06T09:37:46.620197709-07:00" level=info msg="Default bridge (docker0) is assigned with an IP address 172.17.0.0/16. Daemon option --bip

Oct 06 09:37:46 ubuntu dockerd[1310]: time="2017-10-06T09:37:46.656105194-07:00" level=info msg="Loading containers: done."

Oct 06 09:37:46 ubuntu dockerd[1310]: time="2017-10-06T09:37:46.690695015-07:00" level=info msg="Docker daemon" commit=afdb6d4 graphdriver(s)=overlay2 version=17.09.0-ce

Oct 06 09:37:46 ubuntu dockerd[1310]: time="2017-10-06T09:37:46.690941375-07:00" level=info msg="Daemon has completed initialization"

Oct 06 09:37:46 ubuntu systemd[1]: Started Docker Application Container Engine.

Oct 06 09:37:46 ubuntu dockerd[1310]: time="2017-10-06T09:37:46.696875094-07:00" level=info msg="API listen on /var/run/docker.sock"

```
user@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.

```
user@nodea:~$ ls -l /var/run/docker.sock
srw-rw---- 1 root docker 0 Sep 28 19:53 /var/run/docker.sock
user@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:

```
user@nodea:~$ docker version

Client:
 Version:      17.09.0-ce
 API version:  1.32
 Go version:   go1.8.3
 Git commit:   afdb6d4
 Built:        Tue Sep 26 22:42:18 2017
 OS/Arch:      linux/amd64

Server:
 Version:      17.09.0-ce
 API version:  1.32 (minimum version 1.12)
 Go version:   go1.8.3
 Git commit:   afdb6d4
 Built:        Tue Sep 26 22:40:56 2017
 OS/Arch:      linux/amd64
 Experimental: false

user@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 17.06 uses API version 1.30 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:

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

Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
Images: 0
Server Version: 17.09.0-ce
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 logentries splunk syslog
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 06b9cb35161009dcb7123345749fef02f7cea8e0
runc version: 3f2f8b84a77f73d38244dd690525642a72156c64
init version: 949e6fa
Security Options:
  apparmor
  seccomp
   Profile: default
Kernel Version: 4.4.0-96-generic
Operating System: Ubuntu 16.04.1 LTS
OSType: linux
Architecture: x86_64
CPUs: 2
```

```
Total Memory: 1.936GiB
Name: ubuntu
ID: UL4C:FRU3:SL4M:4FXL:TUTN:RTQU:WSF7:WOP6:K3EM:FW72:M67K:LSFQ
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https://index.docker.io/v1/
Experimental: false
Insecure Registries:
  127.0.0.0/8
Live Restore Enabled: false

WARNING: No swap limit support

user@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 (HTTP+JSON)
- 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 we desire, 64-bit Linux and v3.2.6 respectively:

```
user@nodea:~$ wget https://github.com/coreos/etcd/releases/download/v3.2.8/etcd-v3.2.8-linux-amd64.tar.gz
```

```
...  
user@nodea:~$
```

Next extract the files:

```
user@nodea:~$ mkdir etcd  
user@nodea:~$ tar xzf etcd-v3.2.8-linux-amd64.tar.gz  
user@nodea:~$
```

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

```
user@nodea:~$ sudo cp etcd-v3.2.8-linux-amd64/etcd* /usr/bin/  
user@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.

Check your gcc version:

```
user@nodea:~$ gcc --version  
  
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609
```

Copyright (C) 2015 Free Software Foundation, Inc.  
This is free software; see the source for copying conditions. There is NO  
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

```
user@nodea:~$
```

On older systems you can generally install gcc from packages ( `sudo apt-get install gcc` ).

Next download the latest Go binaries, we will use Go v1.9. Go 1.9 came out in April 2017 and Kubernetes 1.7 uses it (the prior version was built using Go v1.8.x)

```
user@nodea:~$ wget https://storage.googleapis.com/golang/go1.9.linux-amd64.tar.gz
```

```
...  
user@nodea:~$
```

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

```
user@nodea:~$ sudo tar -C /usr/local/ -xzf go1.9.linux-amd64.tar.gz
```

```
user@nodea:~$
```

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

```
total 176  
drwxr-xr-x  2 root root  4096 Aug 24 14:50 api  
-rw-r--r--  1 root root 41258 Aug 24 14:50 AUTHORS  
drwxr-xr-x  2 root root  4096 Aug 24 14:51 bin  
drwxr-xr-x  4 root root  4096 Aug 24 14:51 blog  
-rw-r--r--  1 root root  1576 Aug 24 14:50 CONTRIBUTING.md  
-rw-r--r--  1 root root 55577 Aug 24 14:50 CONTRIBUTORS  
drwxr-xr-x  9 root root  4096 Aug 24 14:50 doc  
-rw-r--r--  1 root root  5686 Aug 24 14:50 favicon.ico  
drwxr-xr-x  3 root root  4096 Aug 24 14:50 lib  
-rw-r--r--  1 root root  1479 Aug 24 14:50 LICENSE  
drwxr-xr-x 14 root root  4096 Aug 24 14:51 misc  
-rw-r--r--  1 root root  1303 Aug 24 14:50 PATENTS
```

```
drwxr-xr-x  7 root root  4096 Aug 24 14:51 pkg
-rw-r--r--  1 root root  1601 Aug 24 14:50 README.md
-rw-r--r--  1 root root    26 Aug 24 14:50 robots.txt
drwxr-xr-x 46 root root  4096 Aug 24 14:50 src
drwxr-xr-x 19 root root 12288 Aug 24 14:50 test
-rw-r--r--  1 root root    5 Aug 24 14:50 VERSION
```

```
user@nodea:~$
```

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

```
total 28280
-rwxr-xr-x 1 root root 10369401 Aug 24 14:51 go
-rwxr-xr-x 1 root root 15325248 Aug 24 14:51 godoc
-rwxr-xr-x 1 root root  3257829 Aug 24 14:51 gofmt
```

```
user@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:

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

```
user@ubuntu:~$ 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
```

```
user@nodea:~$
```

Apply the changes by either rebooting or source the change.

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

```
user@nodea:~$ which go
/usr/local/go/bin/go
user@nodea:~$
```

```
user@nodea:~$ go version
go version go1.9 linux/amd64
user@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 v1.7.8:

```
user@nodea:~$ wget https://github.com/kubernetes/kubernetes/archive/v1.7.8.tar.gz
...
user@nodea:~$
```

Next extract the archive to the k8s directory:

```
user@nodea:~$ mkdir k8s

user@nodea:~$ tar xzf v1.7.8.tar.gz -C k8s/ --strip-components=1

user@nodea:~$
```

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

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

total 916
drwxrwxr-x  4 user user  4096 Aug 17 01:29 api
drwxrwxr-x 13 user user  4096 Aug 17 01:29 build
lrwxrwxrwx  1 user user    21 Aug 17 01:29 BUILD.bazel -> build/root/BUILD.root
-rw-rw-r--  1 user user 808363 Aug 17 01:29 CHANGELOG.md
drwxrwxr-x 21 user user  4096 Aug 17 01:29 cluster
drwxrwxr-x 23 user user  4096 Aug 17 01:29 cmd
-rw-rw-r--  1 user user   257 Aug 17 01:29 code-of-conduct.md
-rw-rw-r--  1 user user   290 Aug 17 01:29 CONTRIBUTING.md
drwxrwxr-x 11 user user  4096 Aug 17 01:29 docs
drwxrwxr-x 31 user user  4096 Aug 17 01:29 examples
drwxrwxr-x 13 user user  4096 Aug 17 01:29 federation
drwxrwxr-x  2 user user  4096 Aug 17 01:29 Godeps
drwxrwxr-x 11 user user  4096 Aug 17 01:29 hack
drwxrwxr-x  2 user user  4096 Aug 17 01:29 hooks
-rw-rw-r--  1 user user  6844 Aug 17 01:29 labels.yaml
-rw-rw-r--  1 user user 11358 Aug 17 01:29 LICENSE
drwxrwxr-x  2 user user  4096 Aug 17 01:29 logo
lrwxrwxrwx  1 user user    19 Aug 17 01:29 Makefile -> build/root/Makefile
lrwxrwxrwx  1 user user    35 Aug 17 01:29 Makefile.generated_files -> build/root/Makefile.generated_files
-rw-rw-r--  1 user user   275 Aug 17 01:29 OWNERS
-rw-rw-r--  1 user user  1292 Aug 17 01:29 OWNERS_ALIASES
drwxrwxr-x 41 user user  4096 Aug 17 01:29 pkg
drwxrwxr-x  4 user user  4096 Aug 17 01:29 plugin
-rw-rw-r--  1 user user  3242 Aug 17 01:29 README.md
drwxrwxr-x  3 user user  4096 Aug 17 01:29 staging
drwxrwxr-x 12 user user  4096 Aug 17 01:29 test
drwxrwxr-x  7 user user  4096 Aug 17 01:29 third_party
drwxrwxr-x  4 user user  4096 Aug 17 01:29 translations
-rw-rw-r--  1 user user 11898 Aug 17 01:29 Vagrantfile
```






```
drwxrwxr-x 12 user user 4096 Aug 17 01:29 vendor
lrwxrwxrwx 1 user user 20 Aug 17 01:29 WORKSPACE -> build/root/WORKSPACE

user@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 2GB of RAM. This is not enough to perform the compile. Before you perform the compile increase the memory of your VM to 4GB. You will need to shutdown the machine ( `sudo shutdown -h now` ), change the memory configured and then restart it.

 Ubuntu\_Xenial\_Xerus - VMware Workstation 12 Player (Non-commercial use only)

Player ▾ |     

### Virtual Machine Settings









Hardware Options

Device	Summary
 Memory	3 GB
 Processors	2
 Hard Disk (SCSI)	20 GB
 Network Adapter	NAT
 Display	Auto detect

#### Memory

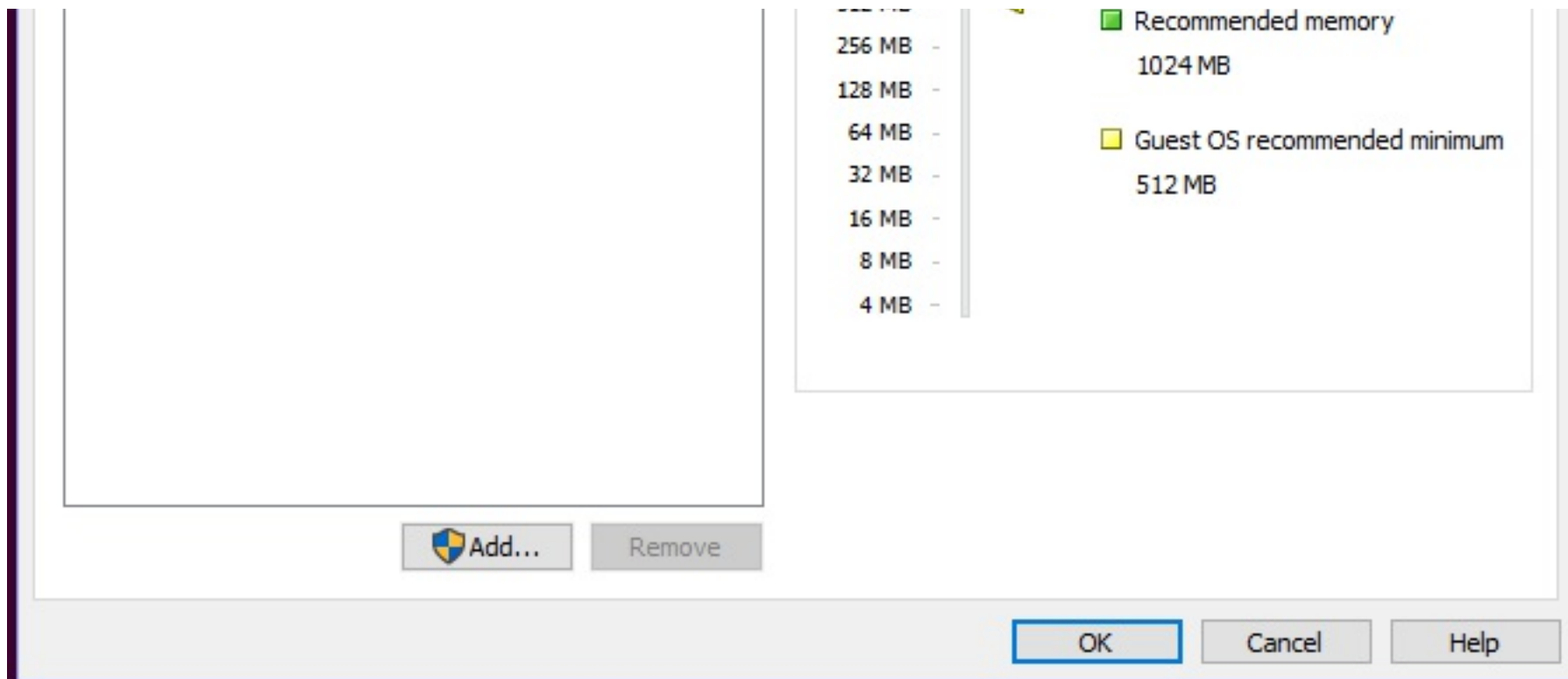
Specify the amount of memory allocated to this virtual machine. The memory size must be a multiple of 4 MB.

Memory for this virtual machine:  MB

64 GB -   
32 GB -   
16 GB -   
8 GB -   
4 GB -   
2 GB -   
1 GB -   
512 MB - 

☒ Maximum recommended memory  
(Memory swapping may occur beyond this size.)  
28628 MB





When the machine has rebooted, log back in. To build the binaries we will first change to a *root* shell and verify that the path includes Go.

```
user@nodea:~$ sudo su -  
root@nodea:~#
```

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

```
root@nodea:/home/user/k8s# which go  
/usr/local/go/bin/go
```

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

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

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

+++ [0828 20:07:58] Building the toolchain targets:
k8s.io/kubernetes/hack/cmd/teststale
k8s.io/kubernetes/vendor/github.com/jteeuwen/go-bindata/go-bindata
+++ [0828 20:07:59] Generating bindata:
test/e2e/generated/gobindata_util.go
/home/user/k8s /home/user/k8s/test/e2e/generated
/home/user/k8s/test/e2e/generated
+++ [0828 20:07:59] Building go targets for linux/amd64:
cmd/libs/go2idl/deepcopy-gen
+++ [0828 20:08:06] Building the toolchain targets:
k8s.io/kubernetes/hack/cmd/teststale
k8s.io/kubernetes/vendor/github.com/jteeuwen/go-bindata/go-bindata
+++ [0828 20:08:06] Generating bindata:
test/e2e/generated/gobindata_util.go
/home/user/k8s /home/user/k8s/test/e2e/generated
/home/user/k8s/test/e2e/generated
+++ [0828 20:08:06] Building go targets for linux/amd64:
cmd/libs/go2idl/defaulters-gen
+++ [0828 20:08:12] Building the toolchain targets:
k8s.io/kubernetes/hack/cmd/teststale
k8s.io/kubernetes/vendor/github.com/jteeuwen/go-bindata/go-bindata
+++ [0828 20:08:12] Generating bindata:
test/e2e/generated/gobindata_util.go
/home/user/k8s /home/user/k8s/test/e2e/generated
/home/user/k8s/test/e2e/generated
+++ [0828 20:08:13] Building go targets for linux/amd64:
cmd/libs/go2idl/conversion-gen
+++ [0828 20:08:19] Building the toolchain targets:
k8s.io/kubernetes/hack/cmd/teststale
k8s.io/kubernetes/vendor/github.com/jteeuwen/go-bindata/go-bindata
+++ [0828 20:08:19] Generating bindata:
test/e2e/generated/gobindata_util.go
/home/user/k8s /home/user/k8s/test/e2e/generated
/home/user/k8s/test/e2e/generated
```

```
+++ [0828 20:08:19] Building go targets for linux/amd64:
cmd/libs/go2idl/openapi-gen
+++ [0828 20:08:27] Building the toolchain targets:
k8s.io/kubernetes/hack/cmd/teststale
k8s.io/kubernetes/vendor/github.com/jteeuwen/go-bindata/go-bindata
+++ [0828 20:08:27] Generating bindata:
test/e2e/generated/gobindata_util.go
/home/user/k8s /home/user/k8s/test/e2e/generated
/home/user/k8s/test/e2e/generated
+++ [0828 20:08:27] 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
vendor/k8s.io/kube-aggregator
vendor/k8s.io/apiextensions-apiserver
plugin/cmd/kube-scheduler
cmd/kubectrl
federation/cmd/kubefed
cmd/gendocs
cmd/genkubedocs
cmd/genman
cmd/genyaml
cmd/mungedocs
cmd/genswaggertypedocs
cmd/linkcheck
federation/cmd/genfeddocs
vendor/github.com/onsi/ginkgo/ginkgo
test/e2e/e2e.test
cmd/kubemark
vendor/github.com/onsi/ginkgo/ginkgo
test/e2e_node/e2e_node.test
cmd/gke-certificates-controller

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

The build will take up to 30 minutes depending on the memory available in your VM. 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 *user* account by entering `exit` in the root shell.

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

logout

user@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:

```
user@nodea:~$ etcd

2017-08-28 20:20:03.603858 I | etcdmain: etcd Version: 3.2.8
2017-08-28 20:20:03.604224 I | etcdmain: Git SHA: 9d43462
2017-08-28 20:20:03.604229 I | etcdmain: Go Version: go1.8.3
2017-08-28 20:20:03.604231 I | etcdmain: Go OS/Arch: linux/amd64
2017-08-28 20:20:03.604235 I | etcdmain: setting maximum number of CPUs to 2, total number of available CPUs is 2
2017-08-28 20:20:03.604243 W | etcdmain: no data-dir provided, using default data-dir ./default.etcd
2017-08-28 20:20:03.607301 I | embed: listening for peers on http://localhost:2380
2017-08-28 20:20:03.607392 I | embed: listening for client requests on localhost:2379
2017-08-28 20:20:03.609886 I | etcdserver: name = default
2017-08-28 20:20:03.609897 I | etcdserver: data dir = default.etcd
2017-08-28 20:20:03.609900 I | etcdserver: member dir = default.etcd/member
2017-08-28 20:20:03.609903 I | etcdserver: heartbeat = 100ms
2017-08-28 20:20:03.609905 I | etcdserver: election = 1000ms
2017-08-28 20:20:03.609907 I | etcdserver: snapshot count = 100000
2017-08-28 20:20:03.609913 I | etcdserver: advertise client URLs = http://localhost:2379
2017-08-28 20:20:03.609918 I | etcdserver: initial advertise peer URLs = http://localhost:2380
2017-08-28 20:20:03.609927 I | etcdserver: initial cluster = default=http://localhost:2380
```

```

2017-08-28 20:20:03.611273 I | etcdserver: starting member 8e9e05c52164694d in cluster cdf818194e3a8c32
2017-08-28 20:20:03.611305 I | raft: 8e9e05c52164694d became follower at term 0
2017-08-28 20:20:03.611316 I | raft: newRaft 8e9e05c52164694d [peers: [], term: 0, commit: 0, applied: 0,
lastindex: 0, lastterm: 0]
2017-08-28 20:20:03.611321 I | raft: 8e9e05c52164694d became follower at term 1
2017-08-28 20:20:03.614557 W | auth: simple token is not cryptographically signed
2017-08-28 20:20:03.615121 I | etcdserver: starting server... [version: 3.2.8, cluster version: to_be_decided]
2017-08-28 20:20:03.616092 I | etcdserver/membership: added member 8e9e05c52164694d [http://localhost:2380] to
cluster cdf818194e3a8c32
2017-08-28 20:20:04.012911 I | raft: 8e9e05c52164694d is starting a new election at term 1
2017-08-28 20:20:04.013660 I | raft: 8e9e05c52164694d became candidate at term 2
2017-08-28 20:20:04.014084 I | raft: 8e9e05c52164694d received MsgVoteResp from 8e9e05c52164694d at term 2
2017-08-28 20:20:04.014249 I | raft: 8e9e05c52164694d became leader at term 2
2017-08-28 20:20:04.014410 I | raft: raft.node: 8e9e05c52164694d elected leader 8e9e05c52164694d at term 2
2017-08-28 20:20:04.014868 I | etcdserver: setting up the initial cluster version to 3.2
2017-08-28 20:20:04.015243 I | etcdserver: published {Name:default ClientURLs:[http://localhost:2379]} to cluster
cdf818194e3a8c32
2017-08-28 20:20:04.015316 N | etcdserver/membership: set the initial cluster version to 3.2
2017-08-28 20:20:04.015386 I | etcdserver/api: enabled capabilities for version 3.2
2017-08-28 20:20:04.015396 I | embed: ready to serve client requests
2017-08-28 20:20:04.016002 E | etcdmain: forgot to set Type=notify in systemd service file?
2017-08-28 20:20:04.022030 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:

```

user@nodea:~$ sudo ~/k8s/_output/bin/kube-apiserver \
--etcd-servers=http://localhost:2379 \
--service-cluster-ip-range=10.0.0.0/16

[sudo] password for user:
I0828 20:21:47.171781 32860 server.go:112] Version: v1.7.8+793658f2d7ca7
I0828 20:21:47.281894 32860 serving.go:279] Generated self-signed cert (/var/run/kubernetes/apiserver.crt,

```

```

/var/run/kubernetes/apiserver.key)
W0828 20:21:47.282577 32860 authentication.go:368] AnonymousAuth is not allowed with the AllowAll authorizer.
Resetting AnonymousAuth to false. You should use a different authorizer
W0828 20:21:47.545928 32860 genericapiserver.go:325] Skipping API autoscaling/v2alpha1 because it has no
resources.
W0828 20:21:47.547169 32860 genericapiserver.go:325] Skipping API batch/v2alpha1 because it has no resources.
[restful] 2017/08/28 20:21:47 log.go:33: [restful/swagger] listing is available at
https://172.16.151.203:6443/swaggerapi
[restful] 2017/08/28 20:21:47 log.go:33: [restful/swagger] https://172.16.151.203:6443/swaggerui/ is mapped to
folder /swagger-ui/
I0828 20:21:51.144367 32860 insecure_handler.go:118] Serving insecurely on 127.0.0.1:8080
I0828 20:21:51.145315 32860 serve.go:85] Serving securely on 0.0.0.0:6443
I0828 20:21:51.145520 32860 crd_finalizer.go:248] Starting CRDFinalizer
I0828 20:21:51.153139 32860 apiservice_controller.go:113] Starting APIServiceRegistrationController
I0828 20:21:51.153174 32860 cache.go:32] Waiting for caches to sync for APIServiceRegistrationController
controller
I0828 20:21:51.153309 32860 available_controller.go:201] Starting AvailableConditionController
I0828 20:21:51.153326 32860 cache.go:32] Waiting for caches to sync for AvailableConditionController controller
I0828 20:21:51.153352 32860 tprregistration_controller.go:144] Starting tpr-autoregister controller
I0828 20:21:51.153358 32860 controller_utils.go:994] Waiting for caches to sync for tpr-autoregister controller
I0828 20:21:51.153508 32860 customresource_discovery_controller.go:152] Starting DiscoveryController
I0828 20:21:51.153932 32860 naming_controller.go:284] Starting NamingConditionController
I0828 20:21:51.170431 32860 autoregister_controller.go:120] Starting autoregister controller
I0828 20:21:51.170535 32860 cache.go:32] Waiting for caches to sync for autoregister controller
I0828 20:21:51.254382 32860 cache.go:39] Caches are synced for APIServiceRegistrationController controller
I0828 20:21:51.254470 32860 cache.go:39] Caches are synced for AvailableConditionController controller
I0828 20:21:51.254488 32860 controller_utils.go:1001] Caches are synced for tpr-autoregister controller
I0828 20:21:51.271838 32860 cache.go:39] Caches are synced for autoregister controller
E0828 20:21:51.351572 32860 autoregister_controller.go:167] v1.authentication.k8s.io failed with : Operation
cannot be fulfilled on apiservices.apiregistration.k8s.io "v1.authentication.k8s.io": the object has been
modified; please apply your changes to the latest version and try again
...

```

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

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

```

{
  "kind": "NodeList",
  "apiVersion": "v1",
  "metadata": {
    "selfLink": "/api/v1/nodes",

```

```
"resourceVersion": "63"
},
"items": []
}

user@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. In a **new terminal** run the kubelet, passing it the IRI for the API Server:

```
user@nodea:~$ sudo ~user/k8s/_output/bin/kubelet --api-servers=http://127.0.0.1:8080
```

```
Flag --api-servers has been deprecated, Use --kubeconfig instead. Will be removed in a future version.
I0828 20:24:15.677441 33074 feature_gate.go:144] feature gates: map[]
W0828 20:24:15.678202 33074 server.go:825] Could not load kubeconfig file /var/lib/kubelet/kubeconfig: stat
/var/lib/kubelet/kubeconfig: no such file or directory. Using default client config instead.
I0828 20:24:15.862031 33074 client.go:72] Connecting to docker on unix:///var/run/docker.sock
I0828 20:24:15.862619 33074 client.go:92] Start docker client with request timeout=2m0s
W0828 20:24:15.917124 33074 cni.go:189] Unable to update cni config: No networks found in /etc/cni/net.d
I0828 20:24:15.926724 33074 manager.go:143] cAdvisor running in container: "/user.slice"
W0828 20:24:16.005061 33074 manager.go:151] unable to connect to Rkt api service: rkt: cannot tcp Dial rkt api
service: dial tcp [::1]:15441: getsockopt: connection refused
I0828 20:24:16.016197 33074 fs.go:117] Filesystem partitions: map[/dev/sda1:{mountpoint:/var/lib/docker/aufs
major:8 minor:1 fsType:ext4 blockSize:0}]
I0828 20:24:16.024865 33074 manager.go:198] Machine: {NumCores:2 CpuFrequency:2711681 MemoryCapacity:4124880896
MachineID:6e883acc04fc7db3713776be57a3dac9 SystemUUID:F2564D56-3460-443D-57E3-836F703215A2 BootID:e21a0ef9-85e0-
4ef1-b0b9-f85c262ea596 Filesystems:[{Device:/dev/sda1 DeviceMajor:8 DeviceMinor:1 Capacity:18889830400 Type:vfs
Inodes:1179648 HasInodes:true}] DiskMap:map[8:0:{Name:sda Major:8 Minor:0 Size:21474836480 Scheduler:deadline}]
NetworkDevices:[{Name:ens33 MacAddress:00:0c:29:32:15:a2 Speed:1000 Mtu:1500}] Topology:[{Id:0 Memory:4124880896
Cores:[{Id:0 Threads:[0] Caches:[{Size:32768 Type:Data Level:1} {Size:32768 Type:Instruction Level:1} {Size:262144
Type:Unified Level:2}]}] Caches:[{Size:8388608 Type:Unified Level:3}]} {Id:2 Memory:0 Cores:[{Id:0 Threads:[1]
Caches:[{Size:32768 Type:Data Level:1} {Size:32768 Type:Instruction Level:1} {Size:262144 Type:Unified Level:2}]}]
Caches:[{Size:8388608 Type:Unified Level:3}]}] CloudProvider:Unknown InstanceType:Unknown InstanceID:None}
I0828 20:24:16.025980 33074 manager.go:204] Version: {KernelVersion:4.4.0-93-generic ContainerOsVersion:Ubuntu
16.04.1 LTS DockerVersion:17.06.2-ce DockerAPIVersion:1.30 CadvisorVersion: CadvisorRevision:}
I0828 20:24:16.027223 33074 server.go:536] --cgroups-per-qos enabled, but --cgroup-root was not specified.
```

```

defaulting to /
W0828 20:24:16.028059    33074 container_manager_linux.go:216] Running with swap on is not supported, please
disable swap! This will be a fatal error by default starting in K8s v1.6! In the meantime, you can opt-in to
making this a fatal error by enabling --experimental-fail-swap-on.
I0828 20:24:16.028214    33074 container_manager_linux.go:246] container manager verified user specified cgroup-
root exists: /
I0828 20:24:16.028254    33074 container_manager_linux.go:251] Creating Container Manager object based on Node
Config: {RuntimeCgroupsName: SystemCgroupsName: KubeletCgroupsName: ContainerRuntime:docker CgroupsPerQOS:true
CgroupRoot:/ CgroupDriver:cgroupfs 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:nodes.available Operator:LessThan Value:{Quantity:<nil> Percentage:0.1}
GracePeriod:0s MinReclaim:<nil>}] {Signal:nodes.inodesFree Operator:LessThan Value:{Quantity:<nil>
Percentage:0.05} GracePeriod:0s MinReclaim:<nil>}}] ExperimentalQOSReserved:map[]}
I0828 20:24:16.029171    33074 kubelet.go:273] Watching apiserver
W0828 20:24:16.036434    33074 kubelet_network.go:70] Hairpin mode set to "promiscuous-bridge" but kubenet is not
enabled, falling back to "hairpin-veth"
I0828 20:24:16.036642    33074 kubelet.go:508] Hairpin mode set to "hairpin-veth"
W0828 20:24:16.063738    33074 cni.go:189] Unable to update cni config: No networks found in /etc/cni/net.d
I0828 20:24:16.070537    33074 docker_service.go:208] Docker cri networking managed by kubernetes.io/no-op
I0828 20:24:16.079891    33074 docker_service.go:225] Setting cgroupDriver to cgroupfs
I0828 20:24:16.096200    33074 remote_runtime.go:42] Connecting to runtime service unix:///var/run/dockershim.sock
I0828 20:24:16.099179    33074 kuberuntime_manager.go:163] Container runtime docker initialized, version: 17.06.2-
ce, apiVersion: 1.30.0
I0828 20:24:16.100568    33074 server.go:943] Started kubelet v1.7.8+793658f2d7ca7
I0828 20:24:16.100979    33074 kubelet_node_status.go:247] Setting node annotation to enable volume controller
attach/detach
E0828 20:24:16.101344    33074 kubelet.go:1229] Image garbage collection failed once. Stats initialization may not
have completed yet: unable to find data for container /
I0828 20:24:16.102989    33074 server.go:132] Starting to listen on 0.0.0.0:10250
I0828 20:24:16.103888    33074 server.go:310] Adding debug handlers to kubelet server.
E0828 20:24:16.109066    33074 kubelet.go:1729] Failed to check if disk space is available for the runtime: failed
to get fs info for "runtime": unable to find data for container /
E0828 20:24:16.109127    33074 kubelet.go:1737] Failed to check if disk space is available on the root partition:
failed to get fs info for "root": unable to find data for container /
I0828 20:24:16.130599    33074 fs_resource_analyzer.go:66] Starting FS ResourceAnalyzer
I0828 20:24:16.131633    33074 status_manager.go:140] Starting to sync pod status with apiserver
I0828 20:24:16.131964    33074 kubelet.go:1809] Starting kubelet main sync loop.
I0828 20:24:16.132108    33074 kubelet.go:1820] skipping pod synchronization - [container runtime is down PLEG is
not healthy: pleg was last seen active 2562047h47m16.854775807s ago; threshold is 3m0s]
W0828 20:24:16.131170    33074 container_manager_linux.go:747] CPUAccounting not enabled for pid: 33074
W0828 20:24:16.132913    33074 container_manager_linux.go:750] MemoryAccounting not enabled for pid: 33074
E0828 20:24:16.131226    33074 container_manager_linux.go:543] [ContainerManager]: Fail to get rootfs information

```



```

unable to find data for container /
I0828 20:24:16.131737 33074 volume_manager.go:245] Starting Kubelet Volume Manager
I0828 20:24:16.152907 33074 factory.go:351] Registering Docker factory
W0828 20:24:16.153226 33074 manager.go:247] Registration of the rkt container factory failed: unable to
communicate with Rkt api service: rkt: cannot tcp Dial rkt api service: dial tcp [::1]:15441: getsockopt:
connection refused
I0828 20:24:16.153433 33074 factory.go:54] Registering systemd factory
I0828 20:24:16.153954 33074 factory.go:86] Registering Raw factory
I0828 20:24:16.154332 33074 manager.go:1121] Started watching for new ooms in manager
I0828 20:24:16.157809 33074 oomparser.go:185] oomparser using systemd
I0828 20:24:16.158817 33074 manager.go:288] Starting recovery of all containers
I0828 20:24:16.235079 33074 kubelet_node_status.go:247] Setting node annotation to enable volume controller
attach/detach
I0828 20:24:16.235325 33074 manager.go:293] Recovery completed
I0828 20:24:16.237160 33074 kubelet_node_status.go:82] Attempting to register node nodea
I0828 20:24:16.243571 33074 kubelet_node_status.go:85] Successfully registered node nodea
E0828 20:24:16.327225 33074 helpers.go:771] Could not find capacity information for resource
storage.kubernetes.io/scratch
W0828 20:24:16.327277 33074 helpers.go:782] eviction manager: no observation found for eviction signal
allocatableNodeFs.available
...

```

In the example above we used the `--api-servers` command line argument, which is now deprecated. In a more robust deployment we would pass the kubelet `--kubeconfig` argument pointing to a configuration file which would contain the IRIs of the API servers. More on the kubeconfig later.

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

- The kubelet searches for the kubeconfig in `/var/lib/kubelet/kubeconfig` by default
- The kubelet automatically connects to docker on the Unix domain socket: `unix:///var/run/docker.sock`
- The kubelet is listening on port 10250
- The kubelet successfully registered node nodea with the API Server

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

```

user@nodea:~$ curl http://localhost:8080/api/v1/nodes && echo
{
  "kind": "NodeList",
  "apiVersion": "v1",
  "metadata": {

```

```

    "selfLink": "/api/v1/nodes",
    "resourceVersion": "81"
  },
  "items": [
    {
      "metadata": {
        "name": "nodea",
        "selfLink": "/api/v1/nodes/nodea",
        "uid": "89616162-8c69-11e7-8c61-000c293215a2",
        "resourceVersion": "81",
        "creationTimestamp": "2017-08-29T03:24:16Z",
        "labels": {
          "beta.kubernetes.io/arch": "amd64",
          "beta.kubernetes.io/os": "linux",
          "kubernetes.io/hostname": "nodea"
        },
        "annotations": {
          "volumes.kubernetes.io/controller-managed-attach-detach": "true"
        }
      },
      "spec": {
        "externalID": "nodea"
      },
      "status": {
        "capacity": {
          "cpu": "2",
          "memory": "4028204Ki",
          "pods": "110"
        },
        "allocatable": {
          "cpu": "2",
          "memory": "3925804Ki",
          "pods": "110"
        },
        "conditions": [
          {
            "type": "OutOfDisk",
            "status": "False",
            "lastHeartbeatTime": "2017-08-29T03:25:26Z",
            "lastTransitionTime": "2017-08-29T03:24:16Z",
            "reason": "KubeletHasSufficientDisk",
            "message": "kubelet has sufficient disk space available"
          }
        ]
      }
    }
  ]
}

```

```

{
  "type": "MemoryPressure",
  "status": "False",
  "lastHeartbeatTime": "2017-08-29T03:25:26Z",
  "lastTransitionTime": "2017-08-29T03:24:16Z",
  "reason": "KubeletHasSufficientMemory",
  "message": "kubelet has sufficient memory available"
},
{
  "type": "DiskPressure",
  "status": "False",
  "lastHeartbeatTime": "2017-08-29T03:25:26Z",
  "lastTransitionTime": "2017-08-29T03:24:16Z",
  "reason": "KubeletHasNoDiskPressure",
  "message": "kubelet has no disk pressure"
},
{
  "type": "Ready",
  "status": "True",
  "lastHeartbeatTime": "2017-08-29T03:25:26Z",
  "lastTransitionTime": "2017-08-29T03:24:26Z",
  "reason": "KubeletReady",
  "message": "kubelet is posting ready status. AppArmor enabled"
}
],
"addresses": [
  {
    "type": "InternalIP",
    "address": "172.16.151.203"
  },
  {
    "type": "Hostname",
    "address": "nodea"
  }
],
"daemonEndpoints": {
  "kubeletEndpoint": {
    "Port": 10250
  }
},
"nodeInfo": {
  "machineID": "6e883acc04fc7db3713776be57a3dac9",
  "systemUUID": "F2564D56-3460-443D-57E3-836F703215A2",

```

```
user@nodea:~$
```

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

NAME	STATUS	AGE	VERSION
nodea	Ready	2m	v1.7.8+793658f2d7ca7

```
user@nodea:~$
```

Display detailed information about *nodea*.

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

```
Name: nodea
Role:
Labels: beta.kubernetes.io/arch=amd64
        beta.kubernetes.io/os=linux
        kubernetes.io/hostname=nodea
Annotations: volumes.kubernetes.io/controller-managed-attach-detach=true
Taints: <none>
CreationTimestamp: Mon, 28 Aug 2017 20:24:16 -0700
Conditions:
  Type                Status  LastHeartbeatTime             LastTransitionTime
Reason
-----
OutOfDisk             False   Mon, 28 Aug 2017 20:27:06 -0700   Mon, 28 Aug 2017 20:24:16 -0700
KubeletHasSufficientDisk kubelet has sufficient disk space available
MemoryPressure        False   Mon, 28 Aug 2017 20:27:06 -0700   Mon, 28 Aug 2017 20:24:16 -0700
KubeletHasSufficientMemory kubelet has sufficient memory available
DiskPressure          False   Mon, 28 Aug 2017 20:27:06 -0700   Mon, 28 Aug 2017 20:24:16 -0700
KubeletHasNoDiskPressure kubelet has no disk pressure
Ready                 True    Mon, 28 Aug 2017 20:27:06 -0700   Mon, 28 Aug 2017 20:24:26 -0700
KubeletReady          kubelet is posting ready status. AppArmor enabled
Addresses:
  InternalIP: 172.16.151.203
  Hostname: nodea
Capacity:
  cpu: 2
  memory: 4028204Ki
  pods: 110
Allocatable:
```

cpu: 2  
memory: 3925804Ki  
pods: 110

System Info:

Machine ID: 6e883acc04fc7db3713776be57a3dac9  
System UUID: F2564D56-3460-443D-57E3-836F703215A2  
Boot ID: e21a0ef9-85e0-4ef1-b0b9-f85c262ea596  
Kernel Version: 4.4.0-93-generic  
OS Image: Ubuntu 16.04.1 LTS  
Operating System: linux  
Architecture: amd64  
Container Runtime Version: docker://Unknown  
Kubelet Version: v1.7.8+793658f2d7ca7  
Kube-Proxy Version: v1.7.8+793658f2d7ca7  
ExternalID: nodea  
Non-terminated Pods: (0 in total)

Namespace	Name	CPU Requests	CPU Limits	Memory Requests	Memory Limits
-----------	------	--------------	------------	-----------------	---------------

Allocated resources:

(Total limits may be over 100 percent, i.e., overcommitted.)

CPU Requests	CPU Limits	Memory Requests	Memory Limits
0 (0%)	0 (0%)	0 (0%)	0 (0%)

Events:

FirstSeen	LastSeen	Count	From	SubObjectPath	Type	Reason
2m	2m	1	kubelet, nodea		Normal	Starting
Starting kubelet.						
2m	2m	2	kubelet, nodea		Normal	NodeHasSufficientDisk
Node nodea status is now: NodeHasSufficientDisk						
2m	2m	2	kubelet, nodea		Normal	NodeHasSufficientMemory
Node nodea status is now: NodeHasSufficientMemory						
2m	2m	2	kubelet, nodea		Normal	NodeHasNoDiskPressure
Node nodea status is now: NodeHasNoDiskPressure						
2m	2m	1	kubelet, nodea		Normal	NodeAllocatableEnforced
Updated Node Allocatable limit across pods						
2m	2m	1	kubelet, nodea		Normal	NodeReady
Node nodea status is now: NodeReady						

user@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.

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

apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  nodeName: nodea
  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: {}

user@nodea:~$
```

Now use `kubectl` to create your pod.

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

```
pod "nginx" created
```

```
user@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):

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

NAME	READY	STATUS	RESTARTS	AGE
nginx	0/2	ContainerCreating	0	15s

```
user@nodea:~$
```

Wait for it!!!

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

NAME	READY	STATUS	RESTARTS	AGE
nginx	2/2	Running	0	33s

```
user@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:

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

```
Name:          nginx
Namespace:     default
Node:          nodea/172.16.151.203
Start Time:    Mon, 28 Aug 2017 20:27:54 -0700
Labels:        <none>
Annotations:   <none>
Status:        Running
IP:            172.17.0.2
Containers:
  nginx:
    Container ID:  docker://f5686fed45515677937a55edbeb51351c40736776a2975568e751b3fd9576c51
    Image:         nginx
    Image ID:      docker-
pullable://nginx@sha256:788fa27763db6d69ad3444e8ba72f947df9e7e163bad7c1f5614f8fd27a311c3
    Port:         80/TCP
    State:         Running
      Started:     Mon, 28 Aug 2017 20:28:08 -0700
    Ready:         True
    Restart Count: 0
    Environment:   <none>
    Mounts:
      /var/log/nginx from nginx-logs (rw)
  log-truncator:
    Container ID:  docker://fde19fc791701823c185f82b9f8b397728dba951e93d4d335ea5eff3949dcc27
    Image:         busybox
    Image ID:      docker-
pullable://busybox@sha256:b82b5740006c1ab823596d2c07f081084ecdb32fd258072707b99f52a3cb8692
    Port:         <none>
    Command:
      /bin/sh
    Args:
      -c
      while true; do cat /dev/null > /logdir/access.log; sleep 10; done
    State:         Running
```

```

    Started:      Mon, 28 Aug 2017 20:28:11 -0700
    Ready:        True
    Restart Count: 0
    Environment:  <none>
    Mounts:
      /logdir from nginx-logs (rw)
Conditions:
  Type           Status
  Initialized    True
  Ready          True
  PodScheduled   True
Volumes:
  nginx-logs:
    Type:        EmptyDir (a temporary directory that shares a pod's lifetime)
    Medium:
QoS Class:      BestEffort
Node-Selectors: <none>
Tolerations:    <none>
Events:
  FirstSeen    LastSeen    Count   From              SubObjectPath               Type            Reason
  Message
  -----
54s          54s          1      kubelet, nodea    MountVolume.SetUp succeeded for volume "nginx-logs"      Normal
SuccessfulMountVolume
52s          52s          1      kubelet, nodea    spec.containers{nginx}      Normal          Pulling
pulling image "nginx"
40s          40s          1      kubelet, nodea    spec.containers{nginx}      Normal          Pulled
Successfully pulled image "nginx"
40s          40s          1      kubelet, nodea    spec.containers{nginx}      Normal          Created
Created container
40s          40s          1      kubelet, nodea    spec.containers{nginx}      Normal          Started
Started container
40s          40s          1      kubelet, nodea    spec.containers{log-truncator} Normal          Pulling
pulling image "busybox"
38s          38s          1      kubelet, nodea    spec.containers{log-truncator} Normal          Pulled
Successfully pulled image "busybox"
38s          38s          1      kubelet, nodea    spec.containers{log-truncator} Normal          Created
Created container
37s          37s          1      kubelet, nodea    spec.containers{log-truncator} Normal          Started
Started container
53s          35s          6      kubelet, nodea                                Warning
MissingClusterDNS      kubelet does not have ClusterDNS IP configured and cannot create Pod using "ClusterFirst"

```

```
policy. Falling back to DNSDefault policy.
```

```
user@nodea:~$
```

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

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

```
IP  
172.17.0.2
```

```
user@nodea:~$
```

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

```
HTTP/1.1 200 OK  
Server: nginx/1.13.3  
Date: Tue, 29 Aug 2017 03:36:08 GMT  
Content-Type: text/html  
Content-Length: 612  
Last-Modified: Tue, 11 Jul 2017 13:06:07 GMT  
Connection: keep-alive  
ETag: "5964cd3f-264"  
Accept-Ranges: bytes
```

```
user@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.

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

```
...  
user@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.

```
user@nodea:~$ curl -s http://localhost:8080/api/v1/pods \
| jq -r ' [.items[].status | to_entries[] | select(.key | endswith("IP"))] | from_entries '
{
  "hostIP": "172.16.151.203",
  "podIP": "172.17.0.2"
}

user@nodea:~$
```

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

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

Name          hostIP          podIP
nginx         172.16.151.203  172.17.0.2

user@nodea:~$
```

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

```
user@nodea:~$ docker network inspect bridge | jq -r ' .[] | select(.Name=="bridge").IPAM.Config[].Subnet '
172.17.0.0/16

user@nodea:~$
```

or via the Docker API and jq:

```
user@nodea:~$ curl -s --unix-socket /var/run/docker.sock http://networks/bridge | jq -r ' .IPAM.Config[].Subnet '
172.17.0.0/16

user@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 successfully built and installed the primary k8s components!

*Copyright (c) 2013-2017 RX-M LLC, Cloud Native Consulting, all rights reserved*