LFS258
Kubernetes Fundamentals

Version 2020-10-19

THE LINUX FOUNDATION

Version 2020-10-19

Contents

List of Figures

Chapter 1

Introduction

1.1   Labs

✎ Exercise 1.1: Configuring the System for sudo

It is very dangerous to run a **root shell** unless absolutely necessary: a single typo or other mistake can cause serious (even fatal) damage.

Thus, the sensible procedure is to configure things such that single commands may be run with superuser privilege, by using the **sudo** mechanism. With **sudo** the user only needs to know their own password and never needs to know the root password.

If you are using a distribution such as **Ubuntu**, you may not need to do this lab to get **sudo** configured properly for the course. However, you should still make sure you understand the procedure.

To check if your system is already configured to let the user account you are using run **sudo**, just do a simple command like:

```
$ sudo ls
```

You should be prompted for your user password and then the command should execute. If instead, you get an error message you need to execute the following procedure.

Launch a root shell by typing **su** and then giving the **root** password, not your user password.

On all recent **Linux** distributions you should navigate to the `/etc/sudoers.d` subdirectory and create a file, usually with the name of the user to whom root wishes to grant **sudo** access. However, this convention is not actually necessary as **sudo** will scan all files in this directory as needed. The file can simply contain:

```
student ALL=(ALL)  ALL
```

if the user is `student`.

An older practice (which certainly still works) is to add such a line at the end of the file `/etc/sudoers`. It is best to do so using the **visudo** program, which is careful about making sure you use the right syntax in your edit.

You probably also need to set proper permissions on the file by typing:

```
$ sudo chmod 440 /etc/sudoers.d/student
```

(Note some **Linux** distributions may require 400 instead of 440 for the permissions.)

After you have done these steps, exit the root shell by typing `exit` and then try to do `sudo ls` again.

There are many other ways an administrator can configure **sudo**, including specifying only certain permissions for certain users, limiting searched paths etc. The `/etc/sudoers` file is very well self-documented.

However, there is one more setting we highly recommend you do, even if your system already has **sudo** configured. Most distributions establish a different path for finding executables for normal users as compared to root users. In particular the directories `/sbin` and `/usr/sbin` are not searched, since **sudo** inherits the `PATH` of the user, not the full root user.

Thus, in this course we would have to be constantly reminding you of the full path to many system administration utilities; any enhancement to security is probably not worth the extra typing and figuring out which directories these programs are in. Consequently, we suggest you add the following line to the `.bashrc` file in your home directory:

```
PATH=$PATH:/usr/sbin:/sbin
```

If you log out and then log in again (you don't have to reboot) this will be fully effective.

Chapter 2

Basics of Kubernetes

2.1   Labs

✎Exercise 2.1: View Online Resources

> **Visit kubernetes.io**
>
> With such a fast changing project, it is important to keep track of updates. The main place to find documentation of the current version is https://kubernetes.io/.

1. Open a browser and visit the https://kubernetes.io/ website.

2. In the upper right hand corner, use the drop down to view the versions available. It will say something like `v1.12`.

3. Select the top level link for `Documentation`. The links on the left of the page can be helpful in navigation.

4. As time permits navigate around other sub-pages such as `SETUP`, `CONCEPTS`, and `TASKS` to become familiar with the layout.

> **Track Kubernetes Issues**
>
> There are hundreds, perhaps thousands, working on Kubernetes every day. With that many people working in parallel there are good resources to see if others are experiencing a similar outage. Both the source code as well as feature and issue tracking are currently on github.com.

1. To view the main page use your browser to visit https://github.com/kubernetes/kubernetes/

2. Click on various sub-directories and view the basic information available.

3. Update your URL to point to https://github.com/kubernetes/kubernetes/issues. You should see a series of issues, feature requests, and support communication.

4. In the search box you probably see some existing text like `isissue is:open:` which allows you to filter on the kind of information you would like to see. Append the search string to read: `isissue is:open label:kind/bug:` then press enter.

5. You should now see bugs in descending date order. Across the top of the issues a menu area allows you to view entries by author, labels, projects, milestones, and assignee as well. Take a moment to view the various other selection criteria.

6. Some times you may want to exclude a kind of output. Update the URL again, but precede the label with a minus sign, like: `is`issue is:open -label:kind/bug:. Now you see everything except bug reports.

Chapter 3

Installation and Configuration

3.1 Labs

✎ Exercise 3.1: Install Kubernetes

Overview

There are several Kubernetes installation tools provided by various vendors. In this lab we will learn to use **kubeadm**. As a community-supported independent tool, it is planned to become the primary manner to build a Kubernetes cluster.

> **ⓘ Platforms: GCP, AWS, VirtualBox, etc**
>
> The labs were written using **Ubuntu** instances running on **G**oogle **C**loud **P**latform (**GCP**). They have been written to be vendor-agnostic so could run on AWS, local hardware, or inside of virtualization to give you the most flexibility and options. Each platform will have different access methods and considerations. As of v1.19.0 the minimum (as in barely works) size for **VirtualBox** is 3vCPU/4G memory/5G minimal OS for `master` and 1vCPU/2G memory/5G minimal OS for worker node. Most other providers work with 2CPU/7.5G.

If using your own equipment you will have to disable swap on every node. There may be other requirements which will be shown as warnings or errors when using the **kubeadm** command. While most commands are run as a regular user, there are some which require root privilege. Please configure **sudo** access as shown in a previous lab. You If you are accessing the nodes remotely, such as with **GCP** or **AWS**, you will need to use an SSH client such as a local terminal or **PuTTY** if not using **Linux** or a Mac. You can download **PuTTY** from www.putty.org. You would also require a `.pem` or `.ppk` file to access the nodes. Each cloud provider will have a process to download or create this file. If attending in-person instructor led training the file will be made available during class.

> **⚠ Very Important**
>
> Please disable any firewalls while learning Kubernetes. While there is a list of required ports for communication between components, the list may not be as complete as necessary. If using **GCP** you can add a rule to the project which `allows all traffic to all ports`. Should you be using **VirtualBox** be aware that inter-VM networking will need to be set to `promiscuous mode`.

In the following exercise we will install Kubernetes on a single node then grow the cluster, adding more compute resources. Both nodes used are the same size, providing 2 vCPUs and 7.5G of memory. Smaller nodes could be used, but would run slower, and may have strange errors.

> **ⓘ YAML files and White Space**
>
> Various exercises will use YAML files, which are included in the text. You are encouraged to write the files when possible, as the syntax of YAML has white space indentation requirements that are important to learn. An important note, **do not** use tabs in your YAML files, **white space only. Indentation matters.**

If using a PDF the use of copy and paste often does not paste the single quote correctly. It pastes as a back-quote instead. You will need to modify it by hand. The files have also been made available as a compressed **tar** file. You can view the resources by navigating to this URL:

https://training.linuxfoundation.org/cm/LFS258

To login use user: `LFtraining` and a password of: `Penguin2014`

Once you find the name and link of the current file, which will change as the course updates, use **wget** to download the file into your node from the command line then expand it like this:

```
$ wget https://training.linuxfoundation.org/cm/LFS258/LFS258_V2020-10-19_SOLUTIONS.tar.bz2 \
           --user=LFtraining --password=Penguin2014
```

```
$ tar -xvf LFS258_V2020-10-19_SOLUTIONS.tar.bz2
```

(**Note**: depending on your PDF viewer, if you are cutting and pasting the above instructions, the underscores may disappear and be replaced by spaces, so you may have to edit the command line by hand!)

> **ⓘ Bionic**
>
> While **Ubuntu 18** `bionic` has become the typical version to deploy, the Kubernetes repository does not yet have matching binaries at the time of this writing. The `xenial` binaries can be used until an update is provided.

Install Kubernetes

> Log into your nodes. If attending in-person instructor led training the node IP addresses will be provided by the instructor. You will need to use a `.pem` or `.ppk` key for access, depending on if you are using **ssh** from a terminal or **PuTTY**. The instructor will provide this to you.

1. Open a terminal session on your first node. For example, connect via **PuTTY** or **SSH** session to the first **GCP** node. The user name may be different than the one shown, `student`. The IP used in the example will be different than the one you will use.

   ```
   [student@laptop ~]$  ssh -i LFS458.pem student@35.226.100.87
   ```

   ```
   1  The authenticity of host '54.214.214.156 (35.226.100.87)' can't be established.
   2  ECDSA key fingerprint is SHA256:IPvznbkx93/Wc+ACwXrCcDDgvBwmvEXC9vmYhk2Wo1E.
   ```

```
3  ECDSA key fingerprint is MD5:d8:c9:4b:b0:b0:82:d3:95:08:08:4a:74:1b:f6:e1:9f.
4  Are you sure you want to continue connecting (yes/no)? yes
5  Warning: Permanently added '35.226.100.87' (ECDSA) to the list of known hosts.
6  <output_omitted>
```

2. Become `root` and update and upgrade the system. You may be asked a few questions. Allow restarts and keep the local version currently installed. Which would be a `yes` then a `2`.

   student@master:~$ sudo -i

   root@master:~# apt-get update && apt-get upgrade -y

```
1  <output_omitted>
2
3  You can choose this option to avoid being prompted; instead,
4  all necessary restarts will be done for you automatically
5  so you can avoid being asked questions on each library upgrade.
```

   Restart services during package upgrades without asking? [yes/no] yes

```
1  <output_omitted>
2
3  A new version (/tmp/fileEbke6q) of configuration file /etc/ssh/sshd_config is
4  available, but the version installed currently has been locally modified.
5
6    1. install the package maintainer's version
7    2. keep the local version currently installed
8    3. show the differences between the versions
9    4. show a side-by-side difference between the versions
10   5. show a 3-way difference between available versions
11   6. do a 3-way merge between available versions
12   7. start a new shell to examine the situation
```

   What do you want to do about modified configuration file sshd_config? 2

```
1  <output_omitted>
```

3. Install a text editor like **nano**, **vim**, or **emacs**. Any will do, the labs use a popular option, **vim**.

   root@master:~# apt-get install -y vim

```
1  <output-omitted>
```

4. The main choices for a container environment are **Docker** and **cri-o**. We suggest **Docker** for class, as **cri-o** is not yet the default when building the cluster with **kubeadm** on Ubuntu.

   The **cri-o** engine is the default in Red Hat products and is being implemented by others. It has not yet gained wide usage in production, but is included here if you want to work with it. Installing **Docker** is a single command. At the moment it takes ten steps to install and configure **crio**

   ⚠️ **Very Important**

   If you want extra challenge use cri-o. Otherwise install Docker

   Please note, install Docker **OR** cri-o. If both are installed the **kubeadm** init process search pattern will use Docker. Also be aware that if you choose to use cri-o you may find encounter different output than shown in the book.

   (a) **If** using Docker:

                             THE LINUX FOUNDATION

```
root@master:~# apt-get install -y docker.io
```

```
1  <output-omitted>
```

(b) **If** using CRI-O:

    i. Use the **modprobe** command to load the `overlay` and the `br_netfilter` modules.

```
root@master:~# modprobe overlay
```

```
root@master:~# modprobe br_netfilter
```

    ii. Create a **sysctl** config file to enable IP forwarding and netfilter settings persistently across reboots.

```
root@master:~# vim /etc/sysctl.d/99-kubernetes-cri.conf
```

```
net.bridge.bridge-nf-call-iptables  = 1
net.ipv4.ip_forward                 = 1
net.bridge.bridge-nf-call-ip6tables = 1
```

    iii. Use the **sysctl** command to apply the config file.

```
root@master:~# sysctl --system
```

```
1  .....
2  * Applying /etc/sysctl.d/99-kubernetes-cri.conf ...
3  net.bridge.bridge-nf-call-iptables = 1
4  net.ipv4.ip_forward = 1
5  net.bridge.bridge-nf-call-ip6tables = 1
6  * Applying /etc/sysctl.d/99-sysctl.conf ...
7  * Applying /etc/sysctl.conf ...
```

    iv. Install a dependent software package.

```
root@master:~# apt-get install -y software-properties-common
```

```
1  <output-omitted>
```

    v. Add the CRI-O software repository. Press ENTER to continue, then update the metadata.

```
root@master:~# add-apt-repository ppa:projectatomic/ppa
```

```
1  Press [ENTER] to continue or Ctrl-c to cancel adding it.
2  ....
```

```
root@master:~# apt-get update
```

    vi. We can now install the cri-o software. Be aware the version may lag behind updates to Kubernetes software.

```
root@master:~# apt-get install -y cri-o-1.15
```

```
1  <output_omitted>
```

    vii. There is a hard coded path for the **conmon** binary which does not match Ubuntu 18.04. Update the `crio.conf` file to use the correct binary path.

```
root@master:~# which conmon
```

```
1  /usr/bin/conmon
```

    viii. Edit the `/etc/crio/crio.conf` file to use the proper binary path. Also configure registries. Unlike Docker we must declare where to find images other than the core Kubernetes images. Be aware this can be done in a few places such as `/etc/containers/registries.d`.

```
root@master:~# vim /etc/crio/crio.conf
```

```
....
# Path to the conmon binary, used for monitoring the OCI runtime.
conmon = "/usr/bin/conmon"          #<-- Edit this line. Around line 91


....

registries = [                      #<-- Edit and add registries. Around line 258
        "docker.io",
        "quay.io",
```

THE
LINUX
FOUNDATION

```
                    "registry.fedoraproject.org",
            ]

            ....
```

ix. Enable cri-o and ensure it is running.

```
root@master:~# systemctl daemon-reload
root@master:~# systemctl enable crio

root@master:~# systemctl start crio
root@master:~# systemctl status crio
```

```
1  crio.service - Container Runtime Interface for OCI (CRI-O)
2    Loaded: loaded (/usr/lib/systemd/system/crio.service; disabled; vendor preset: enabled)
3    Active: active (running) since Mon 2020-02-03 17:00:34 UTC; 7s ago
4      Docs: https://github.com/cri-o/cri-o
5  ....
```

x. Configure **kubelet** to understand how to interact with crio. The following would be **one long line** inside of the file. It is presented here on multiple lines for ease of typing.

```
root@master:~# vim /etc/default/kubelet

KUBELET_EXTRA_ARGS=--feature-gates="AllAlpha=false,RunAsGroup=true"
--container-runtime=remote
--cgroup-driver=systemd
--container-runtime-endpoint='unix:///var/run/crio/crio.sock'
--runtime-request-timeout=5m
```

5. Add a new repo for kubernetes. You could also download a tar file or use code from GitHub. Create the file and add an entry for the main repo for your distribution. We are using the `Ubuntu 18.04` but the `kubernetes-xenial` repo of the software, also include the key word `main`. Note there are four sections to the entry.

```
root@master:~# vim /etc/apt/sources.list.d/kubernetes.list
```

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

6. Add a GPG key for the packages. The command spans three lines. You can omit the backslash when you type. The `OK` is the expected output, not part of the command.

```
root@master:~# curl -s \
    https://packages.cloud.google.com/apt/doc/apt-key.gpg \
    | apt-key add -
```

```
1  OK
```

7. Update with the new repo declared, which will download updated repo information.

```
root@master:~# apt-get update
```

```
1  <output-omitted>
```

8. Install the software. There are regular releases, the newest of which can be used by omitting the equal sign and version information on the command line. Historically new versions have lots of changes and a good chance of a bug or five. As a result we will hold the software at the recent but stable version we install. In a later lab we will update the cluster to a newer version.

```
root@master:~# apt-get install -y \
        kubeadm=1.18.1-00 kubelet=1.18.1-00 kubectl=1.18.1-00
```

```
1  <output-omitted>
```

```
root@master:~# apt-mark hold kubelet kubeadm kubectl
```

```
1  kubelet set on hold.
2  kubeadm set on hold.
3  kubectl set on hold.
```

9. Deciding which pod network to use for Container Networking Interface (**CNI**) should take into account the expected demands on the cluster. There can be only one pod network per cluster, although the **CNI-Genie** project is trying to change this.

    The network must allow container-to-container, pod-to-pod, pod-to-service, and external-to-service communications. As **Docker** uses host-private networking, using the `docker0` virtual bridge and `veth` interfaces would require being on that host to communicate.

    We will use **Calico** as a network plugin which will allow us to use `Network Policies` later in the course. Currently **Calico** does not deploy using CNI by default. Newer versions of **Calico** have included RBAC in the main file. Once downloaded look for the expected IPV4 range for containers to use in the configuration file.

    `root@master:~# wget https://docs.projectcalico.org/manifests/calico.yaml`

10. Use **less** to page through the file. Look for the IPV4 pool assigned to the containers. There are many different configuration settings in this file. Take a moment to view the entire file. The `CALICO_IPV4POOL_CIDR` must match the value given to **kubeadm init** in the following step, whatever the value may be. Avoid conflicts with existing IP ranges of the instance.

    `root@master:~# less calico.yaml`

    calico.yaml
    ```
    1  ....
    2      # The default IPv4 pool to create on startup if none exists. Pod IPs will be
    3      # chosen from this range. Changing this value after installation will have
    4      # no effect. This should fall within `--cluster-cidr`.
    5            - name: CALICO_IPV4POOL_CIDR
    6              value: "192.168.0.0/16"
    7  ....
    ```

11. Find the IP address of the primary interface of the master server. The example below would be the `ens4` interface and an IP of `10.128.0.3`, yours may be different.

    `root@master:~# ip addr show`

    ```
    1  ....
    2  2: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc mq state UP group default qlen 1000
    3      link/ether 42:01:0a:80:00:18 brd ff:ff:ff:ff:ff:ff
    4      inet 10.128.0.3/32 brd 10.128.0.3 scope global ens4
    5         valid_lft forever preferred_lft forever
    6      inet6 fe80::4001:aff:fe80:18/64 scope link
    7         valid_lft forever preferred_lft forever
    8  ....
    ```

12. Add an local DNS alias for our master server. Edit the `/etc/hosts` file and add the above IP address and assign a name `k8smaster`.

    `root@master:~# vim /etc/hosts`

    ```
    10.128.0.3 k8smaster    #<-- Add this line
    127.0.0.1 localhost
    ....
    ```

13. Create a configuration file for the cluster. There are many options we could include, but will only set the control plane endpoint, software version to deploy and podSubnet values. After our cluster is initialized we will view other default values used. Be sure to use the node alias, not the IP so the network certificates will continue to work when we deploy a load balancer in a future lab.

```
root@master:~# vim kubeadm-config.yaml
```

YAML **kubeadm-config.yaml**

```
1  apiVersion: kubeadm.k8s.io/v1beta2
2  kind: ClusterConfiguration
3  kubernetesVersion: 1.18.1               #<-- Use the word stable for newest version
4  controlPlaneEndpoint: "k8smaster:6443"  #<-- Use the node alias not the IP
5  networking:
6    podSubnet: 192.168.0.0/16             #<-- Match the IP range from the Calico config file
```

14. Initialize the master. Read through the output line by line. Expect the output to change as the software matures. At the end are configuration directions to run as a non-root user. The token is mentioned as well. This information can be found later with the **kubeadm token list** command. The output also directs you to create a pod network to the cluster, which will be our next step. Pass the network settings **Calico** has in its configuration file, found in the previous step. **Please note:** the output lists several commands which following exercise steps will complete.

```
root@master:~# kubeadm init --config=kubeadm-config.yaml --upload-certs \
      | tee kubeadm-init.out      # Save output for future review
```

i **Please Note**

What follows is output of **kubeadm init**. Read the next step prior to further typing.

```
1  [init] Using Kubernetes version: v1.18.1
2  [preflight] Running pre-flight checks
3  [WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the
4  Docker cgroup driver. The recommended driver is "systemd".
5
6  ....
7
8  You can now join any number of the control-plane node
9  running the following command on each as root:
10
11  kubeadm join k8smaster:6443 --token vapzqi.et2p9zbkzk29wwth \
12    --discovery-token-ca-cert-hash sha256:f62bf97d4fba6876e4c3ff645df3fca969c06169dee3865aab9d0bca8ec9f8cd \
13    --control-plane --certificate-key 911d41fcada89a18210489afaa036cd8e192b1f122ebb1b79cce1818f642fab8
14
15  Please note that the certificate-key gives access to cluster sensitive
16  data, keep it secret!
17  As a safeguard, uploaded-certs will be deleted in two hours; If
18  necessary, you can use
19  "kubeadm init phase upload-certs --upload-certs" to reload certs afterward.
20
21  Then you can join any number of worker nodes by running the following
22  on each as root:
23
24  kubeadm join k8smaster:6443 --token vapzqi.et2p9zbkzk29wwth \
25    --discovery-token-ca-cert-hash sha256:f62bf97d4fba6876e4c3ff645df3fca969c06169dee3865aab9d0bca8ec9f8cd
26
```

15. As suggested in the directions at the end of the previous output we will allow a `non-root` user admin level access to the cluster. Take a quick look at the configuration file once it has been copied and the permissions fixed.

```
root@master:~# exit
```

```
1  logout
```

```
student@master:~$ mkdir -p $HOME/.kube

student@master:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

student@master:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config

student@master:~$ less .kube/config
```

```
1  apiVersion: v1
2  clusters:
3  - cluster:
4  <output_omitted>
```

16. Apply the network plugin configuration to your cluster. Remember to copy the file to the current, non-root user directory first.

```
student@master:~$ sudo cp /root/calico.yaml .

student@master:~$ kubectl apply -f calico.yaml
```

```
1  configmap/calico-config created
2  customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created
3  customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created
4  customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created
5  <output_omitted>
```

17. While many objects have short names, a **kubectl** command can be a lot to type. We will enable **bash** auto-completion. Begin by adding the settings to the current shell. Then update the $HOME/.bashrc file to make it persistent. Ensure the bash-completion package is installed. If it was not installed, log out then back in for the shell completion to work.

```
student@master:~$ sudo apt-get install bash-completion -y

<exit and log back in>

student@master:~$ source <(kubectl completion bash)

student@master:~$ echo "source <(kubectl completion bash)" >> $HOME/.bashrc
```

18. Test by describing the node again. Type the first three letters of the sub-command then type the **Tab** key. Auto-completion assumes the default namespace. Pass the namespace first to use auto-completion with a different namespace. By pressing **Tab** multiple times you will see a list of possible values. Continue typing until a unique name is used. First look at the current node (your node name may not start with lfs458-), then look at pods in the kube-system namespace. If you see an error instead such as -bash: _get_comp_words_by_ref: command not found revisit the previous step, install the software, log out and back in.

```
student@master:~$ kubectl des<Tab> n<Tab><Tab> lfs458-<Tab>

student@master:~$ kubectl -n kube-s<Tab> g<Tab> po<Tab>
```

19. View other values we could have included in the kubeadm-config.yaml file when creating the cluster.

```
student@master:~$ sudo kubeadm config print init-defaults
```

THE
LINUX
FOUNDATION

```
1  apiVersion: kubeadm.k8s.io/v1beta2
2  bootstrapTokens:
3  - groups:
4    - system:bootstrappers:kubeadm:default-node-token
5    token: abcdef.0123456789abcdef
6    ttl: 24h0m0s
7    usages:
8    - signing
9    - authentication
10 kind: InitConfiguration
11 <output_omitted>
```

✎ Exercise 3.2: Grow the Cluster

> Open another terminal and connect into a your second node. Install **Docker** and Kubernetes software. These are the many, but not all, of the steps we did on the master node.
>
> This book will use the **worker** prompt for the node being added to help keep track of the proper node for each command. Note that the prompt indicates both the user and system upon which run the command.

1. Using the same process as before connect to a second node. If attending an instructor-led class session, use the same `.pem` key and a new IP provided by the instructor to access the new node. Giving a different title or color to the new terminal window is probably a good idea to keep track of the two systems. The prompts can look very similar.

   PLEASE NOTE: If you chose to use **crio** instead of **Docker** as the container engine you should reference the previous portion of the lab for detailed installation steps.

   ```
   student@worker:~$ sudo -i
   ```

   ```
   root@worker:~# apt-get update && apt-get upgrade -y
   ```

   ```
   1  <Again allow services to restart and keep the local version of software>
   ```

   (a) **IF** you chose Docker on the master:
   ```
   root@worker:~# apt-get install -y docker.io
   ```

   (b) **IF** you chose cri-o on the master:
   ```
   See several previous ten steps for cri-o installation details.
   ```

   ```
   root@worker:~# apt-get install -y vim
   ```

   ```
   root@worker:~# vim /etc/apt/sources.list.d/kubernetes.list
   ```

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

   ```
   root@worker:~# curl -s \
           https://packages.cloud.google.com/apt/doc/apt-key.gpg \
           | apt-key add -
   ```

   ```
   root@worker:~# apt-get update
   ```

   ```
   root@worker:~# apt-get install -y \
           kubeadm=1.18.1-00 kubelet=1.18.1-00 kubectl=1.18.1-00
   ```

   ```
   root@worker:~# apt-mark hold kubeadm kubelet kubectl
   ```

2. Find the IP address of your **master** server. The interface name will be different depending on where the node is running. Currently inside of **GCE** the primary interface for this node type is `ens4`. Your interfaces names may be different. From the output we know our master node IP is `10.128.0.3`.

   `student@master:~$ ip addr show ens4 | grep inet`

   ```
   1    inet 10.128.0.3/32 brd 10.128.0.3 scope global ens4
   2    inet6 fe80::4001:aff:fe8e:2/64 scope link
   ```

3. At this point we could copy and paste the **join** command from the master node. That command only works for 2 hours, so we will build our own **join** should we want to add nodes in the future. Find the token on the master node. The token lasts 2 hours by default. If it has been longer, and no token is present you can generate a new one with the **sudo kubeadm token create** command, seen in the following command.

   `student@master:~$ sudo kubeadm token list`

   ```
   1  TOKEN                       TTL    EXPIRES              USAGES...
   2  27eee4.6e66ff60318da929     23h    2017-11-03T13:27:33Z   authe....
   ```

4. We'll assume you are adding a node more than two hours later and create a new token, to use as part of the `join` command.

   `student@master:~$ sudo kubeadm token create`

   ```
   1  27eee4.6e66ff60318da929
   ```

5. Create and use a Discovery Token CA Cert Hash created from the master to ensure the node joins the cluster in a secure manner. Run this on the master node or wherever you have a copy of the CA file. You will get a long string as output.

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

   ```
   1  (stdin)=  6d541678b05652e1fa5d43908e75e67376e994c3483d6683f2a18673e5d2a1b0
   ```

6. On the **worker node** add a local DNS alias for the master server. Edit the `/etc/hosts` file and add the master IP address and assign the name `k8smaster`.

   `root@worker:~# vim /etc/hosts`

   ```
   10.128.0.3 k8smaster     #<-- Add this line
   127.0.0.1 localhost
   ....
   ```

7. Use the token and hash, in this case as `sha256:long-hash` to join the cluster from the **second**/**worker** node. Use the **private** IP address of the master server and port 6443. The output of the **kubeadm init** on the master also has an example to use, should it still be available.

   ```
   root@worker:~# kubeadm join \
       --token 27eee4.6e66ff60318da929 \
       k8smaster:6443 \
       --discovery-token-ca-cert-hash \
       sha256:6d541678b05652e1fa5d43908e75e67376e994c3483d6683f2a18673e5d2a1b0
   ```

   ```
   1  [preflight] Running pre-flight checks
   2      [WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup driver. The recommended \
   3        driver is "systemd". Please follow the guide at https://kubernetes.io/docs/setup/cri/
   4  [preflight] Reading configuration from the cluster...
   5  [preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
   6  [kubelet-start] Downloading configuration for the kubelet from the "kubelet-config-1.15" ConfigMap in the \
   7              kube-system namespace
   8  [kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
   ```

```
 9  [kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
10  [kubelet-start] Activating the kubelet service
11  [kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...
12
13  This node has joined the cluster:
14  * Certificate signing request was sent to apiserver and a response was received.
15  * The Kubelet was informed of the new secure connection details.
16
17  Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

8. Try to run the **kubectl** command on the secondary system. It should fail. You do not have the cluster or authentication keys in your local `.kube/config` file.

   `root@worker:~# exit`

   `student@worker:~$ kubectl get nodes`

   ```
   1  error: no configuration has been provided, try setting
   2  KUBERNETES_MASTER environment variable
   ```

   `student@worker:~$ ls -l .kube`

   ```
   1  ls: cannot access '.kube': No such file or directory
   ```

✏️ Exercise 3.3: Finish Cluster Setup

1. View the available nodes of the cluster. It can take a minute or two for the status to change from `NotReady` to `Ready`. The `NAME` field can be used to look at the details. Your node name will be different.

   `student@master:~$ kubectl get node`

   ```
   1  NAME             STATUS  ROLES    AGE     VERSION
   2  worker     Ready   <none>   50s     v1.18.1
   3  master  Ready   master   28m     v1.18.1
   4
   ```

2. Look at the details of the node. Work line by line to view the resources and their current status. Notice the status of `Taints`. The master won't allow non-infrastructure pods by default for security and resource contention reasons. Take a moment to read each line of output, some appear to be an error until you notice the status shows `False`.

   `student@master:~$ kubectl describe node master`

   ```
   1   Name:                 master
   2   Roles:                master
   3   Labels:               beta.kubernetes.io/arch=amd64
   4                         beta.kubernetes.io/os=linux
   5                         kubernetes.io/hostname=master
   6                         node-role.kubernetes.io/master=
   7   Annotations:          kubeadm.alpha.kubernetes.io/cri-socket: /var/run/dockershim.sock
   8                         node.alpha.kubernetes.io/ttl: 0
   9                         projectcalico.org/IPv4Address: 10.142.0.3/32
   10                        volumes.kubernetes.io/controller-managed-attach-detach: true
   11  CreationTimestamp:    Mon, 07 Jan 2020 22:04:03 +0000
   12  Taints:               node-role.kubernetes.io/master:NoSchedule
   13  <output_omitted>
   ```

3. Allow the master server to run non-infrastructure pods. The master node begins tainted for security and performance reasons. We will allow usage of the node in the training environment, but this step may be skipped in a production environment. Note the **minus sign (-)** at the end, which is the syntax to remove a taint. As the second node does not have the taint you will get a `not found` error.

```
student@master:~$ kubectl describe node | grep -i taint
```

```
1  Taints:                node-role.kubernetes.io/master:NoSchedule
2  Taints:                <none>
```

```
student@master:~$ kubectl taint nodes \
        --all node-role.kubernetes.io/master-
```

```
1
2  node/master untainted
3  error: taint "node-role.kubernetes.io/master:" not found
```

4. Now that the master node is able to execute any pod we **may** find there is a new taint. This behavior began with v1.12.0, requiring a newly added node to be enabled. It has reappeared in versions since then. View, then remove the taint if present. It can take a minute or two for the scheduler to deploy the remaining pods.

```
student@master:~$ kubectl describe node | grep -i taint
```

```
1  Taints:                node.kubernetes.io/not-ready:NoSchedule
2  Taints:                <none>
```

```
student@master:~$ kubectl taint nodes \
        --all node.kubernetes.io/not-ready-
```

```
1
2  node/lfs58-node-1a0a untainted
3  error: taint "node.kubernetes.io/not-ready:" not found
```

5. Determine if the DNS and Calico pods are ready for use. They should all show a status of `Running`. It may take a minute or two to transition from `Pending`.

```
student@master:~$ kubectl get pods --all-namespaces
```

```
1  NAMESPACE       NAME                                    READY    STATUS     RESTARTS   AGE
2  kube-system     calico-node-jlgwr                       1/1      Running    0          6m
3  kube-system     calico-kube-controllers-74b888b647-wlqf5 1/1     Running    0          6m
4  kube-system     calico-node-tpvnr                       2/2      Running    0          6m
5  kube-system     coredns-78fcdf6894-nc5cn                1/1      Running    0          17m
6  kube-system     coredns-78fcdf6894-xs96m                1/1      Running    0          17m
7  <output_omitted>
```

6. **Only if** you notice the `coredns-` pods are stuck in `ContainerCreating` status you may have to delete them, causing new ones to be generated. Delete both pods and check to see they show a `Running` state. Your pod names will be different.

```
student@master:~$ kubectl get pods --all-namespaces
```

```
1  NAMESPACE       NAME                         READY    STATUS             RESTARTS   AGE
2  kube-system     calico-node-qkvzh            2/2      Running            0          59m
3  kube-system     calico-node-vndn7            2/2      Running            0          12m
4  kube-system     coredns-576cbf47c7-rn6v4     0/1      ContainerCreating  0          3s
5  kube-system     coredns-576cbf47c7-vq5dz     0/1      ContainerCreating  0          94m
6  <output_omitted>
```

```
student@master:~$ kubectl -n kube-system delete \
     pod coredns-576cbf47c7-vq5dz coredns-576cbf47c7-rn6v4
```

```
1  pod "coredns-576cbf47c7-vq5dz" deleted
2  pod "coredns-576cbf47c7-rn6v4" deleted
```

7. When it finished you should see a new tunnel, `tunl0`, interface. It may take up to a minute to be created. As you create objects more interfaces will be created, such as `cali` interfaces when you deploy pods, as shown in the output below.

student@master:~$ ip a

```
1  <output_omitted>
2  4: tunl0@NONE: <NOARP,UP,LOWER_UP> mtu 1440 qdisc noqueue state
3  UNKNOWN group default qlen 1000
4      link/ipip 0.0.0.0 brd 0.0.0.0
5      inet 192.168.0.1/32 brd 192.168.0.1 scope global tunl0
6          valid_lft forever preferred_lft forever
7  6: calib0b93ed4661@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu
8  1440 qdisc noqueue state UP group default
9      link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netnsid 1
10     inet6 fe80::ecee:eeff:feee:eeee/64 scope link
11         valid_lft forever preferred_lft forever
12 <output_omitted>
```

✏️ Exercise 3.4: Deploy A Simple Application

We will test to see if we can deploy a simple application, in this case the **nginx** web server.

1. Create a new `deployment`, which is a Kubernetes object, which will deploy an application in a container. Verify it is running and the desired number of containers matches the available.

student@master:~$ kubectl create deployment nginx --image=nginx

```
1  deployment.apps/nginx created
```

student@master:~$ kubectl get deployments

```
1  NAME    READY   UP-TO-DATE   AVAILABLE   AGE
2  nginx   1/1     1            1           8s
```

2. View the details of the deployment. Remember auto-completion will work for sub-commands and resources as well.

student@master:~$ kubectl describe deployment nginx

```
1  Name:                   nginx
2  Namespace:              default
3  CreationTimestamp:      Mon, 23 Apr 2019 22:38:32 +0000
4  Labels:                 app=nginx
5  Annotations:            deployment.kubernetes.io/revision: 1
6  Selector:               app=nginx
7  Replicas:               1 desired | 1 updated | 1 total | 1 ava....
8  StrategyType:           RollingUpdate
9  MinReadySeconds:        0
10 RollingUpdateStrategy:  25% max unavailable, 25% max surge
11 <output_omitted>
```

3. View the basic steps the cluster took in order to pull and deploy the new application. You should see several lines of output. The first column shows the age of each message. Over time older messages will be removed.

student@master:~$ kubectl get events

```
1  <output_omitted>
```

4. You can also view the output in **yaml** format, which could be used to create this deployment again or new deployments. Get the information but change the output to yaml. Note that halfway down there is status information of the current deployment.

student@master:~$ kubectl get deployment nginx -o yaml

```yaml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    annotations:
5      deployment.kubernetes.io/revision: "1"
6    creationTimestamp: 2017-09-27T18:21:25Z
7  <output_omitted>
```

5. Run the command again and redirect the output to a file. Then edit the file. Remove the `creationTimestamp`, `resourceVersion`, `selfLink`, and `uid` lines. Also remove all the lines including and after `status:`, which should be somewhere around line 120, if others have already been removed.

   ```
   student@master:~$ kubectl get deployment nginx -o yaml > first.yaml
   ```

   ```
   student@master:~$ vim first.yaml
   ```

   ```
   1
   2  <Remove the lines mentioned above>
   3
   ```

6. Delete the existing deployment.

   ```
   student@master:~$ kubectl delete deployment nginx
   ```

   ```
   1  deployment.apps "nginx" deleted
   ```

7. Create the deployment again this time using the file.

   ```
   student@master:~$ kubectl create -f first.yaml
   ```

   ```
   1  deployment.apps/nginx created
   ```

8. Look at the yaml output of this iteration and compare it against the first. The creation time stamp, resource version and unique ID we had deleted are in the new file. These are generated for each resource we create, so we need to delete them from yaml files to avoid conflicts or false information. You may notice some time stamp differences as well. The `status` should not be hard-coded either.

   ```
   student@master:~$ kubectl get deployment nginx -o yaml > second.yaml
   ```

   ```
   student@master:~$ diff first.yaml second.yaml
   ```

   ```
   1  <output_omitted>
   ```

9. Now that we have worked with the raw output we will explore two other ways of generating useful YAML or JSON. Use the `--dry-run` option and verify no object was created. Only the prior `nginx` deployment should be found. The output lacks the unique information we removed before, but does have different output such as the `apiVersion`.

   ```
   student@master:~$ kubectl create deployment two --image=nginx --dry-run=client -o yaml
   ```

```yaml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    creationTimestamp: null
5    labels:
6      app: two
7    name: two
8  spec:
9  <output_omitted>
```

```
student@master:~$ kubectl get deployment
```

```
1  NAME    READY   UP-TO-DATE   AVAILABLE   AGE
2  nginx   1/1     1            1           7m
```

10. Existing objects can be viewed in a ready to use YAML output. Take a look at the existing **nginx** deployment.

```
student@master:~$ kubectl get deployments nginx -o yaml
```

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    annotations:
5      deployment.kubernetes.io/revision: "1"
6    creationTimestamp: null
7    generation: 1
8    labels:
9      run: nginx
10  <output_omitted>
```

11. The output can also be viewed in JSON output.

```
student@master:~$ kubectl get deployment nginx -o json
```

```
1  {
2      "apiVersion": "apps/v1",
3      "kind": "Deployment",
4      "metadata": {
5          "annotations": {
6              "deployment.kubernetes.io/revision": "1"
7          },
8  <output_omitted>
```

12. The newly deployed **nginx** container is a light weight web server. We will need to create a `service` to view the default welcome page. Begin by looking at the help output. Note that there are several examples given, about halfway through the output.

```
student@master:~$ kubectl expose -h
```

```
1  <output_omitted>
```

13. Now try to gain access to the web server. As we have not declared a port to use you will receive an error.

```
student@master:~$ kubectl expose deployment/nginx
```

```
1  error: couldn't find port via --port flag or introspection
2  See 'kubectl expose -h' for help and examples.
```

14. To change an object configuration one can use subcommands `apply`, `edit` or `patch` for non-disruptive updates. The `apply` command does a three-way diff of previous, current, and supplied input to determine modifications to make. Fields not mentioned are unaffected. The `edit` function performs a `get`, opens an editor, then an `apply`. You can update API objects in place with JSON patch and merge patch or strategic merge patch functionality.

    If the configuration has resource fields which cannot be updated once initialized then a disruptive update could be done using the `replace --force` option. This deletes first then re-creates a resource.

    Edit the file. Find the container name, somewhere around line 31 and add the port information as shown below.

```
student@master:~$ vim first.yaml
```

```
YAML    first.yaml

1   ....
2      spec:
3         containers:
4         - image: nginx
5           imagePullPolicy: Always
6           name: nginx
7           ports:                          # Add these
8           - containerPort: 80             # three
9             protocol: TCP                 # lines
10          resources: {}
11  ....
```

15. Due to how the object was created we will need to use `replace` to terminate and create a new deployment.

    `student@master:~$ kubectl replace -f first.yaml`

    ```
    1   deployment.apps/nginx replaced
    ```

16. View the Pod and Deployment. Note the `AGE` shows the Pod was re-created.

    `student@master:~$ kubectl get deploy,pod`

    ```
    1   NAME                        READY    UP-TO-DATE    AVAILABLE    AGE
    2   deployment.apps/nginx    1/1      1                1            2m4s
    3
    4   NAME                        READY    STATUS     RESTARTS    AGE
    5   pod/nginx-7db75b8b78-qjffm   1/1      Running    0           8s
    ```

17. Try to expose the resource again. This time it should work.

    `student@master:~$ kubectl expose deployment/nginx`

    ```
    1   service/nginx exposed
    ```

18. Verify the service configuration. First look at the service, then the endpoint information. Note the `ClusterIP` is not the current endpoint. Calico provides the `ClusterIP`. The `Endpoint` is provided by `kubelet` and `kube-proxy`. Take note of the current endpoint IP. In the example below it is `192.168.1.5:80`. We will use this information in a few steps.

    `student@master:~$ kubectl get svc nginx`

    ```
    1   NAME       TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)    AGE
    2   nginx      ClusterIP      10.100.61.122    <none>         80/TCP     3m
    ```

    `student@master:~$ kubectl get ep nginx`

    ```
    1   NAME      ENDPOINTS           AGE
    2   nginx     192.168.1.5:80      26s
    3
    ```

19. Determine which node the container is running on. Log into that node and use **tcpdump**, which you may need to install using **apt-get install**, to view traffic on the `tun0`, as in tunnel zero, interface. The second node in this example. You may also see traffic on an interface which starts with `cali` and some string. Leave that command running while you run **curl** in the following step. You should see several messages go back and forth, including a `HTTP` HTTP/1.1 200 OK: and a `ack` response to the same sequence.

    ```
    student@master:~$ kubectl describe pod nginx-7cbc4b4d9c-d27xw \
                      | grep Node:
    ```

```
1  Node:    worker/10.128.0.5
```

```
student@worker:~$  sudo tcpdump -i tun10
```

```
1  tcpdump: verbose output suppressed, use -v or -vv for full protocol...
2  listening on tun10, link-type EN10MB (Ethernet), capture size...
3  <output_omitted>
```

20. Test access to the Cluster IP, port 80. You should see the generic `nginx` installed and working page. The output should be the same when you look at the `ENDPOINTS` IP address. If the **curl** command times out the pod may be running on the other node. Run the same command on that node and it should work.

```
student@master:~$ curl 10.100.61.122:80
```

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>Welcome to nginx!</title>
5  <style>
6  <output_omitted>
```

```
student@master:~$ curl 192.168.1.5:80
```

21. Now scale up the deployment from one to three web servers.

```
student@master:~$ kubectl get deployment nginx
```

```
1  NAME     READY   UP-TO-DATE    AVAILABLE    AGE
2  nginx    1/1     1             1            12m
```

```
student@master:~$ kubectl scale deployment nginx --replicas=3
```

```
1  deployment.apps/nginx scaled
```

```
student@master:~$ kubectl get deployment nginx
```

```
1  NAME     READY   UP-TO-DATE    AVAILABLE    AGE
2  nginx    3/3     3             3            12m
3
```

22. View the current endpoints. There now should be three. If the `UP-TO-DATE` above said three, but `AVAILABLE` said two wait a few seconds and try again, it could be slow to fully deploy.

```
student@master:~$ kubectl get ep nginx
```

```
1  NAME     ENDPOINTS                                        AGE
2  nginx    192.168.0.3:80,192.168.1.5:80,192.168.1.6:80    7m40s
```

23. Find the oldest pod of the **nginx** deployment and delete it. The `Tab` key can be helpful for the long names. Use the `AGE` field to determine which was running the longest. You may notice activity in the other terminal where **tcpdump** is running, when you delete the pod. The pods with `192.168.0` addresses are probably on the master and the `192.168.1` addresses are probably on the worker

```
student@master:~$ kubectl get pod -o wide
```

```
1  NAME                     READY     STATUS     RESTARTS    AGE     IP
2  nginx-1423793266-7f1qw   1/1       Running    0           14m     192.168.1.5
3  nginx-1423793266-8w2nk   1/1       Running    0           86s     192.168.1.6
4  nginx-1423793266-fbt4b   1/1       Running    0           86s     192.168.0.3
```

V_2020-10-19

```
student@master:~$ kubectl delete pod nginx-1423793266-7f1qw
```

```
1  pod "nginx-1423793266-7f1qw" deleted
```

24. Wait a minute or two then view the pods again. One should be newer than the others. In the following example nine seconds instead of four minutes. If your **tcpdump** was using the `veth` interface of that container it will error out. Also note we are using a short name for the object.

```
student@master:~$  kubectl get po
```

```
1  NAME                     READY     STATUS     RESTARTS    AGE
2  nginx-1423793266-13p69   1/1       Running    0           9s
3  nginx-1423793266-8w2nk   1/1       Running    0           4m1s
4  nginx-1423793266-fbt4b   1/1       Running    0           4m1s
```

25. View the endpoints again. The original endpoint IP is no longer in use. You can delete any of the pods and the `service` will forward traffic to the existing backend pods.

```
student@master:~$ kubectl get ep nginx
```

```
1  NAME      ENDPOINTS                                    AGE
2  nginx     192.168.0.3:80,192.168.1.6:80,192.168.1.7:80  12m
```

26. Test access to the web server again, using the `ClusterIP` address, then any of the endpoint IP addresses. Even though the endpoints have changed you still have access to the web server. This access is only from within the cluster. When done use **ctrl-c** to stop the **tcpdump** command.

```
student@master:~$ curl 10.100.61.122:80
```

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>Welcome to nginx!</title>
5  <style>
6      body
7  <output_omitted>
```

✏️Exercise 3.5: Access from Outside the Cluster

> You can access a Service from outside the cluster using a DNS add-on or environment variables. We will use environment variables to gain access to a Pod.

1. Begin by getting a list of the pods.

```
student@master:~$ kubectl get po
```

```
1  NAME                     READY     STATUS     RESTARTS    AGE
2  nginx-1423793266-13p69   1/1       Running    0           4m10s
3  nginx-1423793266-8w2nk   1/1       Running    0           8m2s
4  nginx-1423793266-fbt4b   1/1       Running    0           8m2s
```

2. Choose one of the pods and use the exec command to run **printenv** inside the pod. The following example uses the first pod listed above.

```
student@master:~$ kubectl exec nginx-1423793266-13p69 \
    -- printenv |grep KUBERNETES
```

```
1  KUBERNETES_SERVICE_PORT=443
2  KUBERNETES_SERVICE_HOST=10.96.0.1
3  KUBERNETES_SERVICE_PORT_HTTPS=443
4  KUBERNETES_PORT=tcp://10.96.0.1:443
5  <output_omitted>
```

3. Find and then delete the existing service for **nginx**.

   `student@master:~$ kubectl get svc`

```
1  NAME          TYPE         CLUSTER-IP       EXTERNAL-IP    PORT(S)    AGE
2  kubernetes    ClusterIP    10.96.0.1        <none>         443/TCP    4h
3  nginx         ClusterIP    10.100.61.122    <none>         80/TCP     17m
```

4. Delete the service.

   `student@master:~$ kubectl delete svc nginx`

```
1  service "nginx" deleted
```

5. Create the service again, but this time pass the `LoadBalancer` type. Check to see the status and note the external ports mentioned. The output will show the `External-IP` as `pending`. Unless a provider responds with a load balancer it will continue to show as pending.

   `student@master:~$ kubectl expose deployment nginx --type=LoadBalancer`

```
1  service/nginx exposed
```

   `student@master:~$ kubectl get svc`

```
1  NAME          TYPE            CLUSTER-IP        EXTERNAL-IP    PORT(S)         AGE
2  kubernetes    ClusterIP       10.96.0.1         <none>         443/TCP         4h
3  nginx         LoadBalancer    10.104.249.102    <pending>      80:32753/TCP    6s
```

6. Open a browser on your local system, not the lab exercise node, and use the public IP of your node and port 32753, shown in the output above. If running the labs on remote nodes like **AWS** or **GCE** use the public IP you used with PuTTY or SSH to gain access.

Figure 3.1: **External Access via Browser**

7. Scale the deployment to zero replicas. Then test the web page again. Once all pods have finished terminating accessing the web page should fail.

   `student@master:~$ kubectl scale deployment nginx --replicas=0`

```
1  deployment.apps/nginx scaled
```

```
student@master:~$ kubectl get po
```

```
1  No resources found in default namespace.
```

8. Scale the deployment up to two replicas. The web page should work again.

```
student@master:~$ kubectl scale deployment nginx --replicas=2
```

```
1  deployment.apps/nginx scaled
```

```
student@master:~$ kubectl get po
```

```
1  NAME                      READY      STATUS     RESTARTS   AGE
2  nginx-1423793266-7x181    1/1        Running    0          6s
3  nginx-1423793266-s6vcz    1/1        Running    0          6s
```

9. Delete the deployment to recover system resources. Note that deleting a `deployment` does not delete the endpoints or services.

```
student@master:~$ kubectl delete deployments nginx
```

```
1  deployment.apps "nginx" deleted
```

```
student@master:~$ kubectl delete ep nginx
```

```
1  endpoints "nginx" deleted
```

```
student@master:~$ kubectl delete svc nginx
```

```
1  service "nginx" deleted
```

Chapter 4

Kubernetes Architecture

4.1 Labs

✎ Exercise 4.1: Basic Node Maintenance

In this section we will backup the **etcd** database then update the version of Kubernetes used on control plane nodes and worker nodes.

Backup The etcd Database

While the upgrade process has become stable, it remains a good idea to backup the cluster state prior to upgrading. There are many tools available in the market to backup and manage etcd, each with a distinct backup and restore process. We will use the included snapshot command, but be aware the exact steps to restore will depend on the tools used, the version of the cluster, and the nature of the disaster being recovered from.

1. Find the data directory of the **etcd** daemon. All of the settings for the pod can be found in the manifest.

   ```
   student@master:~$ sudo grep data-dir /etc/kubernetes/manifests/etcd.yaml
   ```
   ```
   1    - --data-dir=/var/lib/etcd
   ```

2. Log into the **etcd** container and look at the options **etcdctl** provides. Use tab to complete the container name.

   ```
   student@master:~$ kubectl -n kube-system exec -it etcd-<Tab> -- sh
   ```

   On Container

   (a) View the arguments and options to the **etcdctl** command. Take a moment to view the options and arguments available. As the Bourne shell does not have may features it may be easier to copy/paste the majority of the command and arguments after typing them out the first time.

      ```
      # etcdctl -h
      ```
      ```
      1  NAME:
      2       etcdctl - A simple command line client for etcd3.
      ```

25

```
3
4   USAGE:
5           etcdctl [flags]
6   <output_omitted>
```

(b) In order to use TLS, find the three files that need to be passed with the **etcdctl** command. Change into the directory and view available files. Newer versions of **etcd** image have been minimized. As a result you may no longer have the **find** command, or really most commands. One must remember the URL /etc/kubernetes/pki/etcd. As the **ls** command is also missing we can view the files using **echo** instead.

```
# cd /etc/kubernetes/pki/etcd
```

```
# echo *
```

```
1   ca.crt ca.key healthcheck-client.crt healthcheck-client.key
2   peer.crt peer.key server.crt server.key
```

(c) Typing out each of these keys, especially in a locked-down shell can be avoided by using an environmental parameter. Log out of the shell and pass the various paths to the necessary files.

```
# exit
```

3. Check the health of the database using the loopback IP and port 2379. You will need to pass then peer cert and key as well as the Certificate Authority as environmental variables. The command is commented, you do not need to type out the comments or the backslashes.

```
student@master:~$ kubectl -n kube-system exec -it etcd-master -- sh \ #Same as before
 -c "ETCDCTL_API=3 \ #Version to use
 ETCDCTL_CACERT=/etc/kubernetes/pki/etcd/ca.crt \ # Pass the certificate authority
 ETCDCTL_CERT=/etc/kubernetes/pki/etcd/server.crt \ #Pass the peer cert and key
 ETCDCTL_KEY=/etc/kubernetes/pki/etcd/server.key \
 etcdctl endpoint health"   #The command to test the endpoint
```

```
1   https://127.0.0.1:2379 is healthy: successfully committed proposal: took = 11.942936ms
```

4. Determine how many databases are part of the cluster. Three and five are common to provide 50up-arrow to return to the previous command and edit the command without having to type the whole command again.

```
student@master:~$ kubectl -n kube-system exec -it etcd-master -- sh -c \
 "ETCDCTL_API=3 etcdctl --cert=./peer.crt --key=./peer.key --cacert=./ca.crt \
  --endpoints=https://127.0.0.1:2379 member list"
```

```
1   fb50b7ddbf4930ba, started, master, https://10.128.0.35:2380,
2   https://10.128.0.35:2379, false
```

5. You can also view the status of the cluster in a table format.

```
student@master:~$ kubectl -n kube-system exec -it etcd-master -- sh -c "ETCDCTL_API=3 \
 ETCDCTL_CACERT=/etc/kubernetes/pki/etcd/ca.crt ETCDCTL_CERT=/etc/kubernetes/pki/etcd/server.crt \
 ETCDCTL_KEY=/etc/kubernetes/pki/etcd/server.key  etcdctl --endpoints=https://127.0.0.1:2379 \
 -w table endpoint status --cluster"  #<-- Note the addition of -w table
```

```
1   +--------------------------+------------------+---------+---------+-----------+----
2           --------+-----------+------------+--------------------+--------+
3   |         ENDPOINT         |        ID        | VERSION | DB SIZE | IS LEADER |
4           IS LEARNER | RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
5   +--------------------------+------------------+---------+---------+-----------+----
6           --------+-----------+------------+--------------------+--------+
7   | https://10.128.15.193:2379 | 4ef398947c14d552 |  3.4.9  |  4.2 MB |    true   |
```

                   THE LINUX FOUNDATION

```
 8 |            false |              2 |    2315318 |          2315318 |         |
 9 +-------------------------+----------------+--------+---------+----------+----
10 -------+----------+-----------+------------------+-------+
```

6. Now that we know how many etcd databases are in the cluster, and their health, we can back it up. Use the `snapshot` argument to save the snapshot into the container data directory `/var/lib/etcd/`

   ```
   student@master:~$ kubectl -n kube-system exec -it etcd-master -- sh -c "ETCDCTL_API=3 \
     ETCDCTL_CACERT=/etc/kubernetes/pki/etcd/ca.crt ETCDCTL_CERT=/etc/kubernetes/pki/etcd/server.crt \
     ETCDCTL_KEY=/etc/kubernetes/pki/etcd/server.key  etcdctl --endpoints=https://127.0.0.1:2379 \
     snapshot save /var/lib/etcd/snapshot.db
   ```

   ```
    1 {"level":"info","ts":1598380941.6584022,"caller":"snapshot/v3_snapshot.go:110","
    2 msg":"created temporary db file","path":"/var/lib/etcd/snapshot.db.part"}
    3 {"level":"warn","ts":"2020-08-25T18:42:21.671Z","caller":"clientv3/retry_interceptor.go
    4 :116","msg":"retry stream intercept"}
    5 {"level":"info","ts":1598380941.6736135,"caller":"snapshot/v3_snapshot.go:121","
    6 msg":"fetching snapshot","endpoint":"https://127.0.0.1:2379"}
    7 {"level":"info","ts":1598380941.7519674,"caller":"snapshot/v3_snapshot.go:134","
    8 msg":"fetched snapshot","endpoint":"https://127.0.0.1:2379","took":0.093466104}
    9 {"level":"info","ts":1598380941.7521122,"caller":"snapshot/v3_snapshot.go:143","
   10 msg":"saved","path":"/var/lib/etcd/snapshot.db"}
   11 Snapshot saved at /var/lib/etcd/snapshot.db
   ```

7. Verify the snapshot exists from the node perspective, the file date should have been moments earlier.

   ```
   student@master:~$ sudo ls -l /var/lib/etcd/
   ```

   ```
   1 total 3888
   2 drwx------ 4 root root    4096 Aug 25 11:22 member
   3 -rw------- 1 root root 3973152 Aug 25 18:42 snapshot.db
   ```

8. Backup the snapshot as well as other information used to create the cluster both locally as well as another system in case the node becomes unavailable. Remember to create snapshots on a regular basis, perhaps using a cronjob to ensure a timely restore. When using the `snapshot restore` it's important the database not be in use. An HA cluster would remove and replace the control plane node, and not need a restore. More on the restore process can be found here: https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/#restoring-an-etcd-cluster

   ```
   student@master:~$ mkdir $HOME/backup
   student@master:~$ sudo cp /var/lib/etcd/snapshot.db $HOME/backup/snapshot.db-$(date +%m-%d-%y)
   student@master:~$ sudo cp /root/kubeadm-config.yaml $HOME/backup/
   student@master:~$ sudo cp -r /etc/kubernetes/pki/etcd $HOME/backup/
   ```

Upgrade the Cluster

1. Begin by updating the package metadata for **APT**.

   ```
   student@master:~$ sudo apt update
   ```

   ```
   1 Hit:1 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic InRelease
   2 Get:2 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
   3 Get:3 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
   4 Get:5 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
   5 <output_omitted>
   ```

2. View the available packages. The list will be long, you may have to scroll back up to the top to find a recent version. We will choose the `1.19.0` version.

   ```
   student@master:~$ sudo apt-cache madison kubeadm
   ```

```
1      kubeadm |   1.19.0-00 | http://apt.kubernetes.io kubernetes-xenial/main amd64 Packages
2      kubeadm |   1.18.8-00 | http://apt.kubernetes.io kubernetes-xenial/main amd64 Packages
3      kubeadm |   1.18.6-00 | http://apt.kubernetes.io kubernetes-xenial/main amd64 Packages
4      kubeadm |   1.18.5-00 | http://apt.kubernetes.io kubernetes-xenial/main amd64 Packages
5      kubeadm |   1.18.4-01 | http://apt.kubernetes.io kubernetes-xenial/main amd64 Packages
6      kubeadm |   1.18.4-00 | http://apt.kubernetes.io kubernetes-xenial/main amd64 Packages
7      kubeadm |   1.18.3-00 | http://apt.kubernetes.io kubernetes-xenial/main amd64 Packages
8      kubeadm |   1.18.2-00 | http://apt.kubernetes.io kubernetes-xenial/main amd64 Packages
9      kubeadm |   1.18.1-00 | http://apt.kubernetes.io kubernetes-xenial/main amd64 Packages
10     kubeadm |   1.18.0-00 | http://apt.kubernetes.io kubernetes-xenial/main amd64 Packages
11     kubeadm | 1.17.11-00 | http://apt.kubernetes.io kubernetes-xenial/main amd64 Packages
12     kubeadm |   1.17.9-00 | http://apt.kubernetes.io kubernetes-xenial/main amd64 Packages
13  <output_omitted>
```

3. Remove the hold on **kubeadm** and update the package.

   student@master:~$ sudo apt-mark unhold kubeadm

```
1  Canceled hold on kubeadm.
```

   student@master:~$ sudo apt-get install -y kubeadm=1.19.0-00

```
1  Reading package lists... Done
2  Building dependency tree
3  Reading state information... Done
4  <output_omitted>
```

4. Hold the package again to prevent updates along with other software.

   student@master:~$ sudo apt-mark hold kubeadm

```
1  kubeadm set on hold.
```

5. Verify the version of **Kubeadm** installed.

   student@master:~$ sudo kubeadm version

```
1  kubeadm version: &version.Info{Major:"1", Minor:"19", GitVersion:"v1.19.0",
2  GitCommit:"9f2892aab98fe339f3bd70e3c470144299398ace", GitTreeState:"clean",
3  BuildDate:"2020-08-13T16:10:16Z", GoVersion:"go1.13.15", Compiler:"gc",
4  Platform:"linux/amd64"}
```

6. To prepare the master node for update we first need to evict as many pods as possible. The nature of daemonsets is to have them on every node, and some such as `Calico` must remain. Ignore the daemonsets.

   student@master:~$ kubectl drain master --ignore-daemonsets

```
1  node/master cordoned
2  WARNING: ignoring DaemonSet-managed Pods: kube-system/calico-node-r6rkh, kube-system/kube-proxy-kngq6
3  evicting pod kube-system/calico-kube-controllers-5447dc9cbf-6d7nw
4  evicting pod kube-system/coredns-66bff467f8-brl45
5  evicting pod kube-system/coredns-66bff467f8-q5ms8
6  pod/calico-kube-controllers-5447dc9cbf-6d7nw evicted
7  pod/coredns-66bff467f8-brl45 evicted
8  pod/coredns-66bff467f8-q5ms8 evicted
9  node/master evicted
```

7. Use the **upgrade plan** argument to check the existing cluster and then update the software. You may notice that there are versions available later than v1.19.0. Use the v1.19.0 version.

   student@master:~$ sudo kubeadm upgrade plan

```
 1  [upgrade/config] Making sure the configuration is correct:
 2  [upgrade/config] Reading configuration from the cluster...
 3  [upgrade/config] FYI: You can look at this config file with 'kubectl
 4  -n kube-system get cm kubeadm-config -o yaml'
 5  [preflight] Running pre-flight checks.
 6  [upgrade] Running cluster health checks
 7  [upgrade] Fetching available versions to upgrade to
 8  [upgrade/versions] Cluster version: v1.18.1
 9  [upgrade/versions] kubeadm version: v1.19.0
10  [upgrade/versions] Latest stable version: v1.19.0
11  [upgrade/versions] Latest stable version: v1.19.0
12  [upgrade/versions] Latest version in the v1.18 series: v1.18.8
13  [upgrade/versions] Latest version in the v1.18 series: v1.18.8
14
15  Components that must be upgraded manually after you have upgraded the
16  control plane with 'kubeadm upgrade apply':
17  COMPONENT   CURRENT       AVAILABLE
18  kubelet     2 x v1.18.1   v1.18.8
19
20  Upgrade to the latest version in the v1.18 series:
21
22  COMPONENT                 CURRENT   AVAILABLE
23  kube-apiserver            v1.18.1   v1.18.8
24  kube-controller-manager   v1.18.1   v1.18.8
25  kube-scheduler            v1.18.1   v1.18.8
26  kube-proxy                v1.18.1   v1.18.8
27  CoreDNS                   1.6.7     1.7.0
28  etcd                      3.4.3-0   3.4.3-0
29
30  You can now apply the upgrade by executing the following command:
31
32          kubeadm upgrade apply v1.18.8
33
34  _____
35
36  Components that must be upgraded manually after you have upgraded the
37  control plane with 'kubeadm upgrade apply':
38  COMPONENT   CURRENT       AVAILABLE
39  kubelet     2 x v1.18.1   v1.19.0
40
41  Upgrade to the latest stable version:
42
43  COMPONENT                 CURRENT   AVAILABLE
44  kube-apiserver            v1.18.1   v1.19.0
45  kube-controller-manager   v1.18.1   v1.19.0
46  kube-scheduler            v1.18.1   v1.19.0
47  kube-proxy                v1.18.1   v1.19.0
48  CoreDNS                   1.6.7     1.7.0
49  etcd                      3.4.3-0   3.4.9-1
50
51  You can now apply the upgrade by executing the following command:
52
53          kubeadm upgrade apply v1.19.0
54
55  _____
56
57
58  The table below shows the current state of component configs as understood
59  by this version of kubeadm.
60  Configs that have a "yes" mark in the "MANUAL UPGRADE REQUIRED" column require
61  manual config upgrade or resetting to kubeadm defaults before a successful
62  upgrade can be performed. The version to manually upgrade to is denoted in
```

```
63  the "PREFERRED VERSION" column.
64
65  API GROUP                   CURRENT VERSION    PREFERRED VERSION    MANUAL UPGRADE REQUIRED
66  kubeproxy.config.k8s.io     v1alpha1           v1alpha1             no
67  kubelet.config.k8s.io       v1beta1            v1beta1              no
68  ---------------------------------------------------------------
69
```

8. We are now ready to actually upgrade the software. There will be a lot of output. Be aware the command will ask if you want to proceed with the upgrade, answer yes. Take a moment and look for any errors or suggestions, such as upgrading the version of etcd, or some other package.

   `student@master:~$ sudo kubeadm upgrade apply v1.19.0`

```
1   [upgrade/config] Making sure the configuration is correct:
2   [upgrade/config] Reading configuration from the cluster...
3   [upgrade/config] FYI: You can look at this config file with 'kubectl -n
4   kube-system get cm kubeadm-config -oyaml'
5   [preflight] Running pre-flight checks.
6   [upgrade] Running cluster health checks
7   [upgrade/version] You have chosen to change the cluster version to "v1.19.0"
8   [upgrade/versions] Cluster version: v1.18.1
9   [upgrade/versions] kubeadm version: v1.19.0
10  [upgrade/confirm] Are you sure you want to proceed with the upgrade? [y/N]: y    #<--- Answer with y
11
12  <output_omitted>
13
14  [upgrade/successful] SUCCESS! Your cluster was upgraded to "v1.19.0". Enjoy!
15
16  [upgrade/kubelet] Now that your control plane is upgraded, please proceed with
17  upgrading your kubelets if you haven't already done so.
```

9. Check projectcalico.org or other CNI for supported version to match any updates. While the apply command may show some information projects which are not directly connected to Kubernetes may need to be updated to work with the new version.

10. Check the status of the nodes. The master should show scheduling disabled. Also as we have not updated all the software and restarted the daemons it will show the previous version.

    `student@master:~$ kubectl get node`

```
1  NAME       STATUS                    ROLES     AGE      VERSION
2  master     Ready,SchedulingDisabled  master    7h48m    v1.18.1
3  worker     Ready                     <none>    7h46m    v1.18.1
```

11. Release the hold on **kubelet** and **kubectl**.

    `student@master:~$ sudo apt-mark unhold kubelet kubectl`

```
1  Canceled hold on kubelet.
2  Canceled hold on kubectl.
```

12. Upgrade both packages to the same version as **kubeadm**.

    `student@master:~$ sudo apt-get install -y kubelet=1.19.0-00 kubectl=1.19.0-00`

```
1  Reading package lists... Done
2  Building dependency tree
3  Reading state information... Done
4
5  <output_omitted>
6
7  Setting up kubelet (1.19.0-00) ...
8  Setting up kubectl (1.19.0-00) ...
```

13. Again add the hold so other updates don't update the Kubernetes software.

    student@master:~$ sudo apt-mark hold kubelet kubectl

    ```
    1  kubelet set on hold.
    2  kubectl set on hold.
    ```

14. Restart the daemons.

    student@master:~$ sudo systemctl daemon-reload

    student@master:~$ sudo systemctl restart kubelet

15. Verify the master node has been updated to the new version. Then update other master nodes using the same process except **sudo kubeadm upgrade node** instead of **sudo kubeadm upgrade apply**.

    student@master:~$ kubectl get node

    ```
    1  NAME       STATUS                     ROLES     AGE      VERSION
    2  master     Ready,SchedulingDisabled   master    7h50m    v1.19.0
    3  worker     Ready                      <none>    7h48m    v1.18.1
    ```

16. Now make the master available for the scheduler.

    student@master:~$ kubectl uncordon master

    ```
    1  node/m2 uncordoned
    ```

17. Verify the master now shows a `Ready` status.

    student@master:~$ kubectl get node

    ```
    1  NAME       STATUS     ROLES     AGE     VERSION
    2  master     Ready      master    8h      v1.19.0
    3  worker     Ready      <none>    8h      v1.18.1
    ```

18. Now update the worker node(s) of the cluster. **Open a second terminal session to the worker**. Note that you will need to run a couple commands on the master as well, having two sessions open may be helpful. Begin by allowing the software to update on the worker.

    student@worker:~$ sudo apt-mark unhold kubeadm

    ```
    1  Canceled hold on kubeadm.
    ```

19. Update the **kubeadm** package to the same version as the master node.

    student@worker:~$ sudo apt-get update && sudo apt-get install -y kubeadm=1.19.0-00

    ```
    1  Hit:1 http://us-central1.gce.archive.ubuntu.com/ubuntu bionic InRelease
    2
    3  <output_omitted>
    4
    5  Setting up kubeadm (1.19.0-00) ...
    ```

20. Hold the package again.

    student@worker:~$ sudo apt-mark hold kubeadm

    ```
    1  kubeadm set on hold.
    ```

21. Back on the **master terminal session** drain the worker node, but allow the daemonsets to remain.

    student@master:~$ kubectl drain worker --ignore-daemonsets

```
1  node/worker cordoned
2  WARNING: ignoring DaemonSet-managed Pods: kube-system/calico-node-nwm8w, kube-system/kube-proxy-66sh2
3  evicting pod default/before-688b465898-2sdx7
4  evicting pod default/after-6967f9db5-ws852
5  evicting pod kube-system/calico-kube-controllers-5447dc9cbf-5j947
6  <output_omitted>
7
8  pod/coredns-66bff467f8-nxclf evicted
9  pod/calico-kube-controllers-5447dc9cbf-5j947 evicted
10 node/worker evicted
```

22. Return to the **worker** node and download the updated node configuration.

```
student@worker:~$ sudo kubeadm upgrade node
```

```
1  [upgrade] Reading configuration from the cluster...
2  [upgrade] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
3  [upgrade] Skipping phase. Not a control plane node.
4  [kubelet-start] Downloading configuration for the kubelet from the "kubelet-config-1.19" ConfigMap in the kube-system
5  [kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
6  [upgrade] The configuration for this node was successfully updated!
7  [upgrade] Now you should go ahead and upgrade the kubelet package using your package manager.
```

23. Remove the hold on the software then update to the same version as set on the master.

```
student@worker:~$ sudo apt-mark unhold kubelet kubectl
```

```
1  Canceled hold on kubelet.
2  Canceled hold on kubectl.
```

```
student@worker:~$ sudo apt-get install -y kubelet=1.19.0-00 kubectl=1.19.0-00
```

```
1  Reading package lists... Done
2  Building dependency tree
3  Reading state information... Done
4  The following packages were automatically installed and are no longer required:
5
6  <output_omitted>
7
8  Preparing to unpack .../kubectl_1.19.0-00_amd64.deb ...
9  Unpacking kubectl (1.19.0-00) over (1.18.1-00) ...
10 Preparing to unpack .../kubelet_1.19.0-00_amd64.deb ...
11 Unpacking kubelet (1.19.0-00) over (1.18.1-00) ...
12 Setting up kubelet (1.19.0-00) ...
13 Setting up kubectl (1.19.0-00) ...
```

24. Ensure the packages don't get updated when along with regular updates.

```
student@worker:~$ sudo apt-mark hold kubelet kubectl
```

```
1  kubelet set on hold.
2  kubectl set on hold.
```

25. Restart daemon processes for the software to take effect.

```
student@worker:~$ sudo systemctl daemon-reload
```

```
student@worker:~$ sudo systemctl restart kubelet
```

26. Return to the master node. View the status of the nodes. Notice the worker status.

```
student@master:~$  kubectl get node
```

```
1  NAME          STATUS                     ROLES     AGE    VERSION
2  master        Ready                      master    8h     v1.19.0
3  worker        Ready,SchedulingDisabled   <none>    8h     v1.19.0
```

27. Allow pods to be deployed to the worker node.

```
student@master:~$ kubectl uncordon worker
```

```
1  node/worker uncordoned
```

28. Verify the nodes both show a `Ready` status and the same upgraded version.

```
student@master:~$ kubectl get nodes
```

```
1  NAME      STATUS    ROLES     AGE    VERSION
2  master    Ready     master    8h     v1.19.0
3  worker    Ready     <none>    8h     v1.19.0
```

✎ Exercise 4.2: Working with CPU and Memory Constraints

> **Overview**
>
> We will continue working with our cluster, which we built in the previous lab. We will work with `resource limits`, more with `namespaces` and then a complex `deployment` which you can explore to further understand the architecture and relationships.
>
> Use **SSH** or **PuTTY** to connect to the nodes you installed in the previous exercise. We will deploy an application called **stress** inside a container, and then use `resource limits` to constrain the resources the application has access to use.

1. Use a container called `stress`, in a deployment which we will name `hog`, to generate load. Verify you have the container running.

```
student@master:~$ kubectl create deployment hog --image vish/stress
```

```
1  deployment.apps/hog created
```

```
student@master:~$ kubectl get deployments
```

```
1  NAME    READY    UP-TO-DATE    AVAILABLE    AGE
2  hog     1/1      1             1            13s
```

2. Use the `describe` argument to view details, then view the output in `YAML` format. Note there are no settings limiting resource usage. Instead, there are empty curly brackets.

```
student@master:~$ kubectl describe deployment hog
```

```
1  Name:                 hog
2  Namespace:            default
3  CreationTimestamp:    Tue, 08 Jan 2019 17:01:54 +0000
4  Labels:               app=hog
5  Annotations:          deployment.kubernetes.io/revision: 1
6  <output_omitted>
```

```
student@master:~$ kubectl get deployment hog -o yaml
```

```
1  apiVersion: apps/v1
2  kind: Deployment
3  Metadata:
4
5  <output_omitted>
6
7    template:
8      metadata:
9        creationTimestamp: null
10       labels:
11          app: hog
12     spec:
13       containers:
14       - image: vish/stress
15         imagePullPolicy: Always
16         name: stress
17         resources: {}
18         terminationMessagePath: /dev/termination-log
19  <output_omitted>
```

3. We will use the `YAML` output to create our own configuration file.

   `student@master:~$ kubectl get deployment hog -o yaml > hog.yaml`

4. Probably good to remove the `status` output, `creationTimestamp` and other settings. We will also add in `memory` limits found below.

   `student@master:~$ vim hog.yaml`

hog.yaml

```
1   ....
2          imagePullPolicy: Always
3           name: hog
4           resources:                    # Edit to remove {}
5             limits:                     # Add these 4 lines
6               memory: "4Gi"
7             requests:
8               memory: "2500Mi"
9           terminationMessagePath: /dev/termination-log
10          terminationMessagePolicy: File
11  ....
```

5. Replace the deployment using the newly edited file.

   `student@master:~$ kubectl replace -f hog.yaml`

```
1  deployment.apps/hog replaced
```

6. Verify the change has been made. The deployment should now show resource limits.

   `student@master:~$ kubectl get deployment hog -o yaml`

```
1  ....
2          resources:
3            limits:
4              memory: 4Gi
5            requests:
6              memory: 2500Mi
```

```
7          terminationMessagePath: /dev/termination-log
8  ....
```

7. View the `stdio` of the hog container. Note how much memory has been allocated.

```
student@master:~$ kubectl get po
```

```
1  NAME                   READY    STATUS    RESTARTS   AGE
2  hog-64cbfcc7cf-lwq66   1/1      Running   0          2m
```

```
student@master:~$ kubectl logs hog-64cbfcc7cf-lwq66
```

```
1  I1102 16:16:42.638972      1 main.go:26] Allocating "0" memory, in
2     "4Ki" chunks, with a 1ms sleep between allocations
3  I1102 16:16:42.639064      1 main.go:29] Allocated "0" memory
```

8. Open a second and third terminal to access both master and second nodes. Run **top** to view resource usage. You should not see unusual resource usage at this point. The **dockerd** and **top** processes should be using about the same amount of resources. The **stress** command should not be using enough resources to show up.

9. Edit the `hog` configuration file and add arguments for **stress** to consume CPU and memory. The `args:` entry should be indented the same number of spaces as `resources:`.

```
student@master:~$ vim hog.yaml
```

YAML `hog.yaml`

```
1   ....
2          resources:
3            limits:
4              cpu: "1"
5              memory: "4Gi"
6            requests:
7              cpu: "0.5"
8              memory: "500Mi"
9          args:
10         - -cpus
11         - "2"
12         - -mem-total
13         - "950Mi"
14         - -mem-alloc-size
15         - "100Mi"
16         - -mem-alloc-sleep
17         - "1s"
18  ....
```

10. Delete and recreate the deployment. You should see increased CPU usage almost immediately and memory allocation happen in 100M chunks, allocated to the **stress** program via the running **top** command. Check both nodes as the container could deployed to either. Be aware that nodes with a small amount of memory or CPU may encounter issues. Symptoms include master node infrastructure pods failing. Adjust the amount of resources used to allow standard pods to run without error.

```
student@master:~$ kubectl delete deployment hog
```

```
1  deployment.apps "hog" deleted
```

```
student@master:~$ kubectl create -f hog.yaml
```

```
1  deployment.apps/hog created
```

> **i** **Only if top does not show high usage**
>
> Should the resources not show increased use, there may have been an issue inside of the container. Kubernetes may show it as running, but the actual workload has failed. Or the container may have failed; for example if you were missing a parameter the container may `panic`.
>
> `student@master:~$ kubectl get pod`
>
> ```
> 1  NAME                    READY      STATUS     RESTARTS    AGE
> 2  hog-1985182137-5bz2w    0/1        Error      1           5s
> ```
>
> `student@master:~$ kubectl logs hog-1985182137-5bz2w`
>
> ```
> 1  panic: cannot parse '150mi': unable to parse quantity's suffix
> 2
> 3  goroutine 1 [running]:
> 4  panic(0x5ff9a0, 0xc820014cb0)
> 5          /usr/local/go/src/runtime/panic.go:481 +0x3e6
> 6  k8s.io/kubernetes/pkg/api/resource.MustParse(0x7ffe460c0e69, 0x5, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0)
> 7          /usr/local/google/home/vishnuk/go/src/k8s.io/kubernetes/pkg/api/resource/quantity.go:134 +0x287
> 8  main.main()
> 9          /usr/local/google/home/vishnuk/go/src/github.com/vishh/stress/main.go:24 +0x43
> ```
>
> Here is an example of an improper parameter. The container is running, but not allocating memory. It should show the usage requested from the YAML file.
>
> `student@master:~$ kubectl get po`
>
> ```
> 1  NAME                    READY      STATUS     RESTARTS    AGE
> 2  hog-1603763060-x3vnn    1/1        Running    0           8s
> ```
>
> `student@master:~$ kubectl logs hog-1603763060-x3vnn`
>
> ```
> 1  I0927 21:09:23.514921        1 main.go:26] Allocating "0" memory, in "4ki" chunks, with a 1ms sleep \
> 2                                 between allocations
> 3  I0927 21:09:23.514984        1 main.go:39] Spawning a thread to consume CPU
> 4  I0927 21:09:23.514991        1 main.go:39] Spawning a thread to consume CPU
> 5  I0927 21:09:23.514997        1 main.go:29] Allocated "0" memory
> ```

✎ Exercise 4.3: Resource Limits for a Namespace

> The previous steps set limits for that particular `deployment`. You can also set limits on an entire `namespace`. We will create a new `namespace` and configure another `hog` deployment to run within. When set `hog` should not be able to use the previous amount of resources.

1. Begin by creating a new namespace called `low-usage-limit` and verify it exists.

   `student@master:~$ kubectl create namespace low-usage-limit`

   ```
   1  namespace/low-usage-limit created
   ```

   `student@master:~$ kubectl get namespace`

   ```
   1  NAME                STATUS     AGE
   2  default             Active     1h
   3  kube-node-lease     Active     1h
   ```

```
4  kube-public        Active     1h
5  kube-system        Active     1h
6  low-usage-limit    Active     42s
```

2. Create a YAML file which limits CPU and memory usage. The kind to use is `LimitRange`. Remember the file may be found in the example tarball.

   `student@master:~$ vim low-resource-range.yaml`

   low-resource-range.yaml

   ```
   1  apiVersion: v1
   2  kind: LimitRange
   3  metadata:
   4    name: low-resource-range
   5  spec:
   6    limits:
   7    - default:
   8        cpu: 1
   9        memory: 500Mi
   10     defaultRequest:
   11       cpu: 0.5
   12       memory: 100Mi
   13     type: Container
   ```

3. Create the LimitRange object and assign it to the newly created namespace low-usage-limit. You can use `--namespace` or `-n` to declare the namespace.

   `student@master:~$ kubectl --namespace=low-usage-limit \`
   `     create -f low-resource-range.yaml`

   ```
   1  limitrange/low-resource-range created
   ```

4. Verify it works. Remember that every command needs a namespace and context to work. Defaults are used if not provided.

   `student@master:~$ kubectl get LimitRange`

   ```
   1  No resources found in default namespace.
   ```

   `student@master:~$ kubectl get LimitRange --all-namespaces`

   ```
   1  NAMESPACE          NAME                  CREATED AT
   2  low-usage-limit    low-resource-range    2019-01-08T17:54:22
   ```

5. Create a new deployment in the namespace.

   `student@master:~$ kubectl -n low-usage-limit \`
   `     create deployment limited-hog --image vish/stress`

   ```
   1  deployment.apps/limited-hog created
   ```

6. List the current deployments. Note `hog` continues to run in the default namespace. If you chose to use the **Calico** network policy you may see a couple more than what is listed below.

   `student@master:~$ kubectl get deployments --all-namespaces`

```
1  NAMESPACE        NAME                    READY  UP-TO-DATE  AVAILABLE  AGE
2  default          hog                     1/1    1           1          7m57s
3  kube-system      calico-kube-controllers 1/1    1           1          2d10h
4  kube-system      coredns                 2/2    2           2          2d10h
5  low-usage-limit  limited-hog             0/1    0           0          9s
```

7. View all pods within the namespace. Remember you can use the **tab** key to complete the namespace. You may want to type the namespace first so that tab-completion is appropriate to that namespace instead of the `default` namespace.

    `student@master:~$ kubectl -n low-usage-limit get pods`

```
1  NAME                         READY    STATUS    RESTARTS  AGE
2  limited-hog-2556092078-wnpnv 1/1      Running   0         2m11s
```

8. Look at the details of the pod. You will note it has the settings inherited from the entire namespace. The use of shell completion should work if you declare the namespace first.

    ```
    student@lfs459-node-1a0a:~$ kubectl -n low-usage-limit \
        get pod limited-hog-2556092078-wnpnv -o yaml
    ```

```
1  <output_omitted>
2  spec:
3    containers:
4    - image: vish/stress
5      imagePullPolicy: Always
6      name: stress
7      resources:
8        limits:
9          cpu: "1"
10         memory: 500Mi
11       requests:
12         cpu: 500m
13         memory: 100Mi
14     terminationMessagePath: /dev/termination-log
15 <output_omitted>
```

9. Copy and edit the config file for the original `hog` file. Add the namespace: line so that a new deployment would be in the `low-usage-limit` namespace. Delete the `selflink` line.

    `student@master:~$ cp hog.yaml hog2.yaml`

    `student@master:~$ vim hog2.yaml`

YAML hog2.yaml

```
1  ....
2    labels:
3      app: hog
4    name: hog
5    namespace: low-usage-limit      #<<--- Add this line, delete following
6    selfLink: /apis/apps/v1/namespaces/default/deployments/hog
7  spec:
8  ....
```

10. Open up extra terminal sessions so you can have **top** running in each. When the new deployment is created it will probably be scheduled on the node not yet under any stress.

    Create the deployment.

```
student@master:~$ kubectl create -f hog2.yaml
```

```
1  deployment.apps/hog created
```

11. View the deployments. Note there are two with the same name, `hog` but in different namespaces. You may also find the `calico-typha` deployment has no pods, nor has any requested. Our small cluster does not need to add **Calico** pods via this autoscaler.

```
student@master:~$ kubectl get deployments --all-namespaces
```

```
1  NAMESPACE        NAME                     READY  UP-TO-DATE  AVAILABLE  AGE
2  default          hog                      1/1    1           1          24m
3  kube-system      calico-kube-controllers  1/1    0           0          4h
4  kube-system      coredns                  2/2    2           2          4h
5  low-usage-limit  hog                      1/1    1           1          26s
6  low-usage-limit  limited-hog              1/1    1           1          5m11s
```

12. Look at the **top** output running in other terminals. You should find that both `hog` deployments are using about the same amount of resources, once the memory is fully allocated. Per-deployment settings override the global namespace settings. You should see something like the following lines one from each node, which indicates use of one processor and about 12 percent of your memory, were you on a system with 8G total.

```
1  25128 root      20   0  958532 954672   3180 R 100.0 11.7   0:52.27 stress
2  24875 root      20   0  958532 954800   3180 R 100.3 11.7  41:04.97 stress
```

13. Delete the `hog` deployments to recover system resources.

```
student@master:~$ kubectl -n low-usage-limit delete deployment hog
```

```
1  deployment.apps "hog" deleted
```

```
student@master:~$ kubectl delete deployment hog
```

```
1  deployment.apps "hog" deleted
```

Chapter 5

APIs and Access

5.1   Labs

✎Exercise 5.1: Configuring TLS Access

> **Overview**
>
> Using the Kubernetes API, **kubectl** makes API calls for you. With the appropriate TLS keys you could run **curl** as well use a **golang** client. Calls to the `kube-apiserver` get or set a PodSpec, or desired state. If the request represents a new state the **Kubernetes Control Plane** will update the cluster until the current state matches the specified state. Some end states may require multiple requests. For example, to delete a `ReplicaSet`, you would first set the number of replicas to zero, then delete the `ReplicaSet`.
>
> An API request must pass information as JSON. **kubectl** converts `.yaml` to JSON when making an API request on your behalf. The API request has many settings, but must include `apiVersion`, `kind` and `metadata`, and `spec` settings to declare what kind of container to deploy. The `spec` fields depend on the object being created.
>
> We will begin by configuring remote access to the `kube-apiserver` then explore more of the API.

1. Begin by reviewing the **kubectl** configuration file. We will use the three certificates and the API server address.

   `student@master:~$ less $HOME/.kube/config`

   ```
   1  <output_omitted>
   ```

2. We will create a variables using certificate information. You may want to double-check each parameter as you set it. Begin with setting the `client-certificate-data` key.

   `student@master:~$ export client=$(grep client-cert $HOME/.kube/config |cut -d" " -f 6)`

   `student@master:~$ echo $client`

   ```
   1  LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUM4akNDQWRxZ0F3SUJ
   2  BZOlJRy9wbC9rWEpNdmd3RFFZSktvWklodmNOQVFFTEJRQXdGVEVUTUJFRO
   3  ExVUUKQXhNS2EzVmlaWEp1WlhSbGN6QWVGdzB4TnpFeU1UXhOelEyTXpKY
   4  UZ3MHhPREV5TVRNeE56UTJNelJhTURReApGekFGWQmdOVkJBb1REbk41YzNS
   5  <output_omitted>
   ```

3. Almost the same command, but this time collect the `client-key-data` as the `key` variable.

```
student@master:~$ export key=$(grep client-key-data $HOME/.kube/config |cut -d " " -f 6)

student@master:~$ echo $key
```

```
1  <output_omitted>
```

4. Finally set the `auth` variable with the `certificate-authority-data` key.

```
student@master:~$ export auth=$(grep certificate-authority-data $HOME/.kube/config |cut -d " " -f 6)

student@master:~$ echo $auth
```

```
1  <output_omitted>
```

5. Now encode the keys for use with **curl**.

```
student@master:~$ echo $client | base64 -d - > ./client.pem

student@master:~$ echo $key | base64 -d - > ./client-key.pem

student@master:~$ echo $auth | base64 -d - > ./ca.pem
```

6. Pull the API server URL from the config file. Your hostname or IP address may be different.

```
student@master:~$ kubectl config view |grep server
```

```
1      server: https://k8smaster:6443
```

7. Use **curl** command and the encoded keys to connect to the API server. Use `your` hostname, or IP, found in the previous command, which may be different than the example below.

```
student@master:~$ curl --cert ./client.pem \
    --key ./client-key.pem \
    --cacert ./ca.pem \
    https://k8smaster:6443/api/v1/pods

{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "selfLink": "/api/v1/pods",
    "resourceVersion": "239414"
  },
<output_omitted>
```

8. If the previous command was successful, create a JSON file to create a new pod. Remember to use **find** and search for this file in the tarball output, it can save you some typing.

```
student@master:~$ vim curlpod.json

{
    "kind": "Pod",
    "apiVersion": "v1",
    "metadata":{
        "name": "curlpod",
        "namespace": "default",
        "labels": {
            "name": "examplepod"
        }
    },
    "spec": {
```

```
            "containers": [{
                "name": "nginx",
                "image": "nginx",
                "ports": [{"containerPort": 80}]
            }]
        }
    }
```

9. The previous **curl** command can be used to build a `XPOST` API call. There will be a lot of output, including the scheduler and taints involved. Read through the output. In the last few lines the phase will probably show `Pending`, as it's near the beginning of the creation process.

```
student@master:~$ curl --cert ./client.pem \
    --key ./client-key.pem --cacert ./ca.pem \
    https://k8smaster:6443/api/v1/namespaces/default/pods \
    -XPOST -H'Content-Type: application/json' \
    -d@curlpod.json


{
  "kind": "Pod",
  "apiVersion": "v1",
  "metadata": {
      "name": "curlpod",
<output_omitted>
```

10. Verify the new pod exists and shows a `Running` status.

```
student@master:~$ kubectl get pods
```

```
1  NAME         READY      STATUS     RESTARTS   AGE
2  curlpod      1/1        Running    0          45s
```

✎ Exercise 5.2: Explore API Calls

1. One way to view what a command does on your behalf is to use **strace**. In this case, we will look for the current endpoints, or targets of our API calls. Install the tool, if not present.

```
student@master:~$ sudo apt-get install -y strace
```

```
student@master:~$ kubectl get endpoints
```

```
1  NAME            ENDPOINTS            AGE
2  kubernetes      10.128.0.3:6443     3h
```

2. Run this command again, preceded by **strace**. You will get a lot of output. Near the end you will note several **openat** functions to a local directory, `/home/student/.kube/cache/discovery/k8smaster_6443`. If you cannot find the lines, you may want to redirect all output to a file and `grep` for them. This information is cached, so you may see some differences should you run the command multiple times. As well your IP address may be different.

```
student@master:~$ strace kubectl get endpoints
```

```
1  execve("/usr/bin/kubectl", ["kubectl", "get", "endpoints"], [/*....
2  ....
3  openat(AT_FDCWD, "/home/student/.kube/cache/discovery/k8smaster_6443..
4  <output_omitted>
```

3. Change to the parent directory and explore. Your endpoint IP will be different, so replace the following with one suited to your system.

```
student@master:~$ cd /home/student/.kube/cache/discovery/
```

```
student@master:~/.kube/cache/discovery$ ls
```

```
1  k8smaster_6443
```

```
student@master:~/.kube/cache/discovery$ cd k8smaster_6443/
```

4. View the contents. You will find there are directories with various configuration information for kubernetes.

```
student@master:~/.kube/cache/discovery/k8smaster_6443$ ls
```

```
1  admissionregistration.k8s.io  batch               node.k8s.io
2  apiextensions.k8s.io          certificates.k8s.io policy
3  apiregistration.k8s.io        coordination.k8s.io rbac.authorization.k8s.io
4  apps                          crd.projectcalico.org scheduling.k8s.io
5  authorization.k8s.io          extensions          storage.k8s.io
6  authentication.k8s.io         events.k8s.io       servergroups.json
7  autoscaling                   networking.k8s.io   v1
```

5. Use the find command to list out the subfiles. The prompt has been modified to look better on this page.

```
student@master:./k8smaster_6443$ find .
```

```
1   .
2   ./extensions
3   ./extensions/v1beta1
4   ./extensions/v1beta1/serverresources.json
5   ./node.k8s.io
6   ./node.k8s.io/v1beta1
7   ./node.k8s.io/v1beta1/serverresources.json
8   ./networking.k8s.io
9   ./networking.k8s.io/v1
10  ./networking.k8s.io/v1/serverresources.json
11  ./networking.k8s.io/v1beta1
12  ./networking.k8s.io/v1beta1/serverresources.json
13  <output_omitted>
```

6. View the objects available in version 1 of the API. For each object, or kind:, you can view the verbs or actions for that object, such as create seen in the following example. Note the prompt has been truncated for the command to fit on one line. Some are HTTP verbs, such as GET, others are product specific options, not standard HTTP verbs. The command may be **python**, depending on what version is installed.

```
student@master:.$ python3 -m json.tool v1/serverresources.json
```

```
{
    "apiVersion": "v1",
    "groupVersion": "v1",
    "kind": "APIResourceList",
    "resources": [
        {
            "kind": "Binding",
            "name": "bindings",
            "namespaced": true,
            "singularName": "",
            "verbs": [
                "create"
            ]
        },
    <output_omitted>
```

7. Some of the objects have shortNames, which makes using them on the command line much easier. Locate the shortName for endpoints.

```
student@master:.$ python3 -m json.tool v1/serverresources.json | less
```

```
JSON    serverresources.json

 1  ....
 2  {
 3  "kind": "Endpoints",
 4  "name": "endpoints",
 5  "namespaced": true,
 6  "shortNames": [
 7  "ep"
 8  ],
 9  "singularName": "",
10  "verbs": [
11  "create",
12  "delete",
13  ....
```

8. Use the `shortName` to view the endpoints. It should match the output from the previous command.

   ```
   student@master:.$ kubectl get ep
   ```

   ```
   1  NAME          ENDPOINTS          AGE
   2  kubernetes    10.128.0.3:6443    3h
   ```

9. We can see there are 37 objects in version 1 file.

   ```
   student@master:.$ python3 -m json.tool v1/serverresources.json | grep kind
   ```

   ```
   1      "kind": "APIResourceList",
   2              "kind": "Binding",
   3              "kind": "ComponentStatus",
   4              "kind": "ConfigMap",
   5              "kind": "Endpoints",
   6              "kind": "Event",
   7  <output_omitted>
   ```

10. Looking at another file we find nine more.

    ```
    student@master:$ python3 -m json.tool \
              apps/v1/serverresources.json | grep kind
    ```

    ```
    1      "kind": "APIResourceList",
    2              "kind": "ControllerRevision",
    3              "kind": "DaemonSet",
    4              "kind": "DaemonSet",
    5              "kind": "Deployment",
    6  <output_omitted>
    ```

11. Delete the `curlpod` to recoup system resources.a

    ```
    student@master:$ kubectl delete po curlpod
    ```

    ```
    1  pod "curlpod" deleted
    ```

12. Take a look around the other files in this directory as time permits.

THE LINUX FOUNDATION

Chapter 6

API Objects

6.1 Labs

✎ Exercise 6.1: RESTful API Access

> **Overview**
>
> We will continue to explore ways of accessing the control plane of our cluster. In the security chapter we will discuss there are several authentication methods, one of which is use of a `Bearer token` We will work with one then deploy a local proxy server for application-level access to the Kubernetes API.

We will use the **curl** command to make API requests to the cluster, in an insecure manner. Once we know the IP address and port, then the token we can retrieve cluster data in a RESTful manner. By default most of the information is restricted, but changes to authentication policy could allow more access.

1. First we need to know the IP and port of a node running a replica of the API server. The master system will typically have one running. Use **kubectl config view** to get overall cluster configuration, and find the server entry. This will give us both the IP and the port.

   student@master:~$ kubectl config view

   ```
   1  apiVersion: v1
   2  clusters:
   3  - cluster:
   4      certificate-authority-data: REDACTED
   5      server: https://k8smaster:6443
   6    name: kubernetes
   7  <output_omitted>
   ```

2. Next we need to find the bearer token. This is part of a default token. Look at a list of tokens, first all on the cluster, then just those in the default namespace. There will be a `secret` for each of the controllers of the cluster.

   student@master:~$ kubectl get secrets --all-namespaces

```
1  NAMESPACE     NAME                    TYPE                               ...
2  default       default-token-jdqp7   kubernetes.io/service-account-token...
3  kube-public  default-token-b2prn    kubernetes.io/service-account-token...
4  kube-system  attachdetach-controller-token-ckwvh kubernetes.io/servic...
5  kube-system  bootstrap-signer-token-wpx66 kubernetes.io/service-accou...
6  <output_omitted>
```

```
student@master:~$ kubectl get secrets
```

```
1  NAME                  TYPE                                 DATA     AGE
2  default-token-jdqp7  kubernetes.io/service-account-token  3        23h
```

3. Look at the details of the secret. We will need the `token:` information from the output.

```
student@master:~$ kubectl describe secret default-token-jdqp7
```

```
1  Name:         default-token-jdqp7
2  Namespace:    default
3  Labels:       <none>
4  <output_omitted>
5  token:        eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJrdWJlcm5ldGVz
6  L3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3Bh
7  Y2UiOiJkZWZhdWx0Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZWNyZXQubm
8  <output_omitted>
```

4. Using your mouse to cut and paste, or **cut**, or **awk** to save the data, from the first character `eyJh` to the last, to a variable named `token`. Your token data will be different.

```
student@master:~$ export token=$(kubectl describe \
    secret default-token-jdqp7 |grep ^token |cut -f7 -d ' ')
```

5. Test to see if you can get basic API information from your cluster. We will pass it the server name and port, the token and use the **-k** option to avoid using a cert.

```
student@master:~$ curl https://k8smaster:6443/apis \
        --header "Authorization: Bearer $token" -k

{
  "kind": "APIGroupList",
  "apiVersion": "v1",
  "groups": [
    {
      "name": "apiregistration.k8s.io",
      "versions": [
        {
          "groupVersion": "apiregistration.k8s.io/v1",
          "version": "v1"
<output_omitted>
```

6. Try the same command, but look at API v1. Note that the path has changed to `api`.

```
student@master:~$ curl https://k8smaster:6443/api/v1 \
        --header "Authorization: Bearer $token" -k
```

```
1  <output_omitted>
```

7. Now try to get a list of namespaces. This should return an error. It shows our request is being seen as `system`serviceaccount:, which does not have the `RBAC` authorization to list all namespaces in the cluster.

```
student@master:~$ curl \
        https://k8smaster:6443/api/v1/namespaces \
        --header "Authorization: Bearer $token" -k
```

```
1  <output_omitted>
2    "message": "namespaces is forbidden: User \"system:serviceaccount:default...
3  <output_omitted>
```

8. Pods can also make use of included certificates to use the API. The certificates are automatically made available to a pod under the `/var/run/secrets/kubernetes.io/serviceaccount/`. We will deploy a simple Pod and view the resources. If you view the `token` file you will find it is the same value we put into the `$token` variable. The **-i** will request a **-t** terminal session of the `busybox` container. Once you exit the container will not restart and the pod will show as completed.

```
student@master:~$ kubectl run -i -t busybox --image=busybox --restart=Never
```

Inside container

```
# ls /var/run/secrets/kubernetes.io/serviceaccount/
ca.crt namespace token
# exit
```

9. Clean up by deleting the `busybox` container.

```
student@master:~$ kubectl delete pod busybox
```

```
1  pod "busybox" deleted
```

✎ Exercise 6.2: Using the Proxy

Another way to interact with the API is via a `proxy`. The proxy can be run from a node or from within a `Pod` through the use of a sidecar. In the following steps we will deploy a proxy listening to the loopback address. We will use **curl** to access the API server. If the **curl** request works, but does not from outside the cluster, we have narrowed down the issue to authentication and authorization instead of issues further along the API ingestion process.

1. Begin by starting the proxy. It will start in the foreground by default. There are several options you could pass. Begin by reviewing the help output.

```
student@master:~$ kubectl proxy -h
```

```
1  Creates a proxy server or application-level gateway between localhost
2  and the Kubernetes API Server. It also allows serving static content
3  over specified HTTP path. All incoming data enters through one port
4  and gets forwarded to the remote kubernetes API Server port, except
5  for the path matching the static content path.
6
7  Examples:
8    # To proxy all of the kubernetes api and nothing else, use:
9
10   $ kubectl proxy --api-prefix=/
11 <output_omitted>
```

2. Start the proxy while setting the API prefix, and put it in the background. You may need to use `enter` to view the prompt. Take note of the process ID, 225000 in the example below, we'll use it to kill the process when we are done.

```
student@master:~$ kubectl proxy --api-prefix=/ &
```

```
1  [1] 22500
2  Starting to serve on 127.0.0.1:8001
```

3. Now use the same **curl** command, but point toward the IP and port shown by the proxy. The output should be the same as without the proxy, but may be formatted differently.

   student@master:~$ curl http://127.0.0.1:8001/api/

   ```
   1  <output_omitted>
   ```

4. Make an API call to retrieve the namespaces. The command did not work in the previous section due to permissions, but should work now as the `proxy` is making the request on your behalf.

   student@master:~$ curl http://127.0.0.1:8001/api/v1/namespaces

   ```
   {
     "kind": "NamespaceList",
     "apiVersion": "v1",
     "metadata": {
       "selfLink": "/api/v1/namespaces",
       "resourceVersion": "86902"
   <output_omitted>
   ```

5. Stop the proxy service as we won't need it any more. Use the process ID from a previous step. Your process ID may be different.

   student@master:~$ kill 22500

✎ Exercise 6.3: Working with Jobs

While most API objects are deployed such that they continue to be available there are some which we may want to run a particular number of times called a `Job`, and others on a regular basis called a `CronJob`

Create A Job

1. Create a job which will run a container which sleeps for three seconds then stops.

   student@master:~$ vim job.yaml

   job.yaml

   ```
   1  apiVersion: batch/v1
   2  kind: Job
   3  metadata:
   4    name: sleepy
   5  spec:
   6    template:
   7      spec:
   8        containers:
   9        - name: resting
   10         image: busybox
   11         command: ["/bin/sleep"]
   12         args: ["3"]
   13       restartPolicy: Never
   ```

2. Create the job, then verify and view the details. The example shows checking the job three seconds in and then again after it has completed. You may see different output depending on how fast you type.

   student@master:~$ kubectl create -f job.yaml

   ```
   1  job.batch/sleepy created
   ```

```
student@master:~$ kubectl get job
```

```
1  NAME       COMPLETIONS   DURATION   AGE
2  sleepy     0/1                      3s         3s
```

```
student@master:~$ kubectl describe jobs.batch sleepy
```

```
1   Name:           sleepy
2   Namespace:      default
3   Selector:       controller-uid=24c91245-d0fb-11e8-947a-42010a800002
4   Labels:         controller-uid=24c91245-d0fb-11e8-947a-42010a800002
5                   job-name=sleepy
6   Annotations:    <none>
7   Parallelism:    1
8   Completions:    1
9   Start Time:     Tue, 16 Oct 2018 04:22:50 +0000
10  Completed At:   Tue, 16 Oct 2018 04:22:55 +0000
11  Duration:       5s
12  Pods Statuses:  0 Running / 1 Succeeded / 0 Failed
13  <output_omitted>
```

```
student@master:~$ kubectl get job
```

```
1  NAME       COMPLETIONS   DURATION   AGE
2  sleepy     1/1                      5s         17s
```

3. View the configuration information of the job. There are three parameters we can use to affect how the job runs. Use **-o yaml** to see these parameters. We can see that `backoffLimit`, `completions`, and the `parallelism`. We'll add these parameters next.

```
student@master:~$ kubectl get jobs.batch sleepy -o yaml
```

```
1  <output_omitted>
2    uid: c2c3a80d-d0fc-11e8-947a-42010a800002
3  spec:
4    backoffLimit: 6
5    completions: 1
6    parallelism: 1
7    selector:
8      matchLabels:
9  <output_omitted>
```

4. As the job continues to `AGE` in a completion state, delete the job.

```
student@master:~$ kubectl delete jobs.batch sleepy
```

```
1  job.batch "sleepy" deleted
```

5. Edit the YAML and add the `completions:` parameter and set it to **5**.

```
student@master:~$ vim job.yaml
```

job.yaml
```
1  <output_omitted>
2  metadata:
3    name: sleepy
4  spec:
5    completions: 5   #<--Add this line
```

```yaml
6    template:
7      spec:
8        containers:
9  <output_omitted>
```

6. Create the job again. As you view the job note that `COMPLETIONS` begins as zero of **5**.

   `student@master:~$ kubectl create -f job.yaml`

   ```
   1  job.batch/sleepy created
   ```

   `student@master:~$ kubectl get jobs.batch`

   ```
   1  NAME       COMPLETIONS   DURATION    AGE
   2  sleepy     0/5           5s          5s
   ```

7. View the pods that running. Again the output may be different depending on the speed of typing.

   `student@master:~$ kubectl get pods`

   ```
   1  NAME                  READY    STATUS        RESTARTS    AGE
   2  sleepy-z5tnh          0/1      Completed     0           8s
   3  sleepy-zd692          1/1      Running       0           3s
   4  <output_omitted>
   ```

8. Eventually all the jobs will have completed. Verify then delete the job.

   `student@master:~$ kubectl get jobs`

   ```
   1  NAME       COMPLETIONS   DURATION    AGE
   2  sleepy     5/5           26s         10m
   ```

   `student@master:~$ kubectl delete jobs.batch sleepy`

   ```
   1  job.batch "sleepy" deleted
   ```

9. Edit the YAML again. This time add in the `parallelism:` parameter. Set it to **2** such that two pods at a time will be deployed.

   `student@master:~$ vim job.yaml`

job.yaml

```yaml
1  <output_omitted>
2    name: sleepy
3  spec:
4    completions: 5
5    parallelism: 2    #<-- Add this line
6    template:
7       spec:
8  <output_omitted>
```

10. Create the `job` again. You should see the pods deployed two at a time until all five have completed.

    `student@master:~$ kubectl create -f job.yaml`

```
1  job.batch/sleepy created
```

student@master:~$ kubectl get pods

```
1  NAME                      READY    STATUS     RESTARTS    AGE
2  sleepy-8xwpc              1/1      Running    0           5s
3  sleepy-xjqnf             1/1      Running    0           5s
4  <output_omitted>
```

student@master:~$ kubectl get jobs

```
1  NAME       COMPLETIONS    DURATION    AGE
2  sleepy     3/5            11s         11s
```

11. Add a parameter which will stop the job after a certain number of seconds. Set the `activeDeadlineSeconds:` to 15. The job and all pods will end once it runs for 15 seconds. We will also increase the sleep argument to five, just to be sure does not expire by itself.

    student@master:~$ vim job.yaml

```
1  <output_omitted>
2     completions: 5
3     parallelism: 2
4     activeDeadlineSeconds: 15    #<-- Add this line
5     template:
6       spec:
7         containers:
8         - name: resting
9           image: busybox
10          command: ["/bin/sleep"]
11          args: ["5"]              #<-- Edit this line
12  <output_omitted>
```

12. Delete and recreate the job again. It should run for 15 seconds, usually 3/5, then continue to age without further completions.

    student@master:~$ kubectl delete jobs.batch sleepy

```
1  job.batch "sleepy" deleted
```

    student@master:~$ kubectl create -f job.yaml

```
1  job.batch/sleepy created
```

    student@master:~$ kubectl get jobs

```
1  NAME       COMPLETIONS    DURATION    AGE
2  sleepy     1/5            6s          6s
```

    student@master:~$ kubectl get jobs

```
1  NAME       COMPLETIONS    DURATION    AGE
2  sleepy     3/5            16s         16s
```

13. View the `message:` entry in the `Status` section of the object YAML output.

```
student@master:~$ kubectl get job sleepy -o yaml
```

```
1  <output_omitted>
2  status:
3    conditions:
4    - lastProbeTime: 2018-10-16T05:45:14Z
5      lastTransitionTime: 2018-10-16T05:45:14Z
6      message: Job was active longer than specified deadline
7      reason: DeadlineExceeded
8      status: "True"
9      type: Failed
10   failed: 2
11   startTime: 2018-10-16T05:44:59Z
12   succeeded: 3
```

14. Delete the job.

```
student@master:~$ kubectl delete jobs.batch sleepy
```

```
1  job.batch "sleepy" deleted
```

Create a CronJob

A `CronJob` creates a watch loop which will create a batch job on your behalf when the time becomes true. We Will use our existing `Job` file to start.

1. Copy the `Job` file to a new file.

```
student@master:~$ cp job.yaml cronjob.yaml
```

2. Edit the file to look like the annotated file shown below. Edit the lines mentioned below. The three parameters we added will need to be removed. Other lines will need to be further indented.

```
student@master:~$ vim cronjob.yaml
```

```
1  apiVersion: batch/v1beta1     #<-- Add beta1 to be v1beta1
2  kind: CronJob                 #<-- Update this line to CronJob
3  metadata:
4    name: sleepy
5  spec:
6    schedule: "*/2 * * * *"     #<-- Add Linux style cronjob syntax
7    jobTemplate:                #<-- New jobTemplate and spec move
8      spec:
9        template:               #<-- This and following lines move
10         spec:                 #<-- four spaces to the right
11           containers:
12           - name: resting
13             image: busybox
14             command: ["/bin/sleep"]
15             args: ["5"]
16           restartPolicy: Never
```

3. Create the new `CronJob`. View the jobs. It will take two minutes for the `CronJob` to run and generate a new batch `Job`.

```
student@master:~$ kubectl create -f cronjob.yaml
```

```
1  cronjob.batch/sleepy created
```

```
student@master:~$ kubectl get cronjobs.batch
```

```
1  NAME     SCHEDULE      SUSPEND   ACTIVE   LAST SCHEDULE   AGE
2  sleepy   */2 * * * *   False     0        <none>          8s
```

```
student@master:~$ kubectl get jobs.batch
```

```
1  No resources found.
```

4. After two minutes you should see jobs start to run.

```
student@master:~$ kubectl get cronjobs.batch
```

```
1  NAME     SCHEDULE      SUSPEND   ACTIVE   LAST SCHEDULE   AGE
2  sleepy   */2 * * * *   False     0        21s             2m1s
```

```
student@master:~$ kubectl get jobs.batch
```

```
1  NAME                COMPLETIONS   DURATION   AGE
2  sleepy-1539722040   1/1           5s         18s
```

```
student@master:~$ kubectl get jobs.batch
```

```
1  NAME                COMPLETIONS   DURATION   AGE
2  sleepy-1539722040   1/1           5s         5m17s
3  sleepy-1539722160   1/1           6s         3m17s
4  sleepy-1539722280   1/1           6s         77s
```

5. Ensure that if the job continues for more than 10 seconds it is terminated. We will first edit the **sleep** command to run for 30 seconds then add the `activeDeadlineSeconds:` entry to the container.

```
student@master:~$ vim cronjob.yaml
```

```
1   ....
2     jobTemplate:
3       spec:
4         template:
5           spec:
6             activeDeadlineSeconds: 10   #<-- Add this line
7             containers:
8             - name: resting
9   ....
10          command: ["/bin/sleep"]
11          args: ["30"]                  #<-- Edit this line
12        restartPolicy: Never
13  ....
```

6. Delete and recreate the `CronJob`. It may take a couple of minutes for the batch `Job` to be created and terminate due to the timer.

```
student@master:~$ kubectl delete cronjobs.batch sleepy
```

```
1  cronjob.batch "sleepy" deleted
```

```
student@master:~$ kubectl create -f cronjob.yaml
```

```
1  cronjob.batch/sleepy created
```

```
student@master:~$ kubectl get jobs
```

```
1  NAME                 COMPLETIONS   DURATION    AGE
2  sleepy-1539723240    0/1           61s         61s
```

```
student@master:~$ kubectl get cronjobs.batch
```

```
1  NAME      SCHEDULE      SUSPEND    ACTIVE    LAST SCHEDULE    AGE
2  sleepy    */2 * * * *   False      1         72s              94s
```

```
student@master:~$ kubectl get jobs
```

```
1  NAME                 COMPLETIONS   DURATION    AGE
2  sleepy-1539723240    0/1           75s         75s
```

```
student@master:~$ kubectl get jobs
```

```
1  NAME                 COMPLETIONS   DURATION    AGE
2  sleepy-1539723240    0/1           2m19s       2m19s
3  sleepy-1539723360    0/1           19s         19s
```

```
student@master:~$ kubectl get cronjobs.batch
```

```
1  NAME      SCHEDULE      SUSPEND    ACTIVE    LAST SCHEDULE    AGE
2  sleepy    */2 * * * *   False      2         31s              2m53s
```

7. Clean up by deleting the `CronJob`.

```
student@master:~$  kubectl delete cronjobs.batch sleepy
```

```
1  cronjob.batch "sleepy" deleted
```

Chapter 7

Managing State With Deployments

7.1 Labs

✏ Exercise 7.1: Working with ReplicaSets

> **Overview**
>
> Understanding and managing the state of containers is a core Kubernetes task. In this lab we will first explore the API objects used to manage groups of containers. The objects available have changed as Kubernetes has matured, so the Kubernetes version in use will determine which are available. Our first object will be a `ReplicaSet`, which does not include newer management features found with `Deployments`. A `Deployment` operator manages `ReplicaSet` operators for you. We will also work with another object and watch loop called a `DaemonSet` which ensures a container is running on newly added node.
>
> Then we will update the software in a container, view the revision history, and roll-back to a previous version.

A `ReplicaSet` is a next-generation of a `Replication Controller`, which differs only in the selectors supported. The only reason to use a `ReplicaSet` anymore is if you have no need for updating container software or require update orchestration which won't work with the typical process.

1. View any current `ReplicaSets`. If you deleted resources at the end of a previous lab, you should have none reported in the `default` namespace.

   student@master:~$ `kubectl get rs`

   ```
   1  No resources found in default namespace.
   ```

2. Create a YAML file for a simple `ReplicaSet`. The `apiVersion` setting depends on the version of Kubernetes you are using. The object is stable using the `apps/v1` apiVersion. We will use an older version of **nginx** then update to a newer version later in the exercise.

   student@master:~$ `vim rs.yaml`

rs.yaml

```yaml
 1  apiVersion: apps/v1
 2  kind: ReplicaSet
 3  metadata:
 4    name: rs-one
 5  spec:
 6    replicas: 2
 7    selector:
 8      matchLabels:
 9        system: ReplicaOne
10    template:
11      metadata:
12        labels:
13          system: ReplicaOne
14      spec:
15        containers:
16        - name: nginx
17          image: nginx:1.15.1
18          ports:
19          - containerPort: 80
```

3. Create the `ReplicaSet`:

   `student@master:~$ kubectl create -f rs.yaml`

   ```
   1  replicaset.apps/rs-one created
   ```

4. View the newly created `ReplicaSet`:

   `student@master:~$ kubectl describe rs rs-one`

   ```
   1  Name:           rs-one
   2  Namespace:      default
   3  Selector:       system=ReplicaOne
   4  Labels:         <none>
   5  Annotations:    <none>
   6  Replicas:       2 current / 2 desired
   7  Pods Status:    2 Running / 0 Waiting / 0 Succeeded / 0 Failed
   8  Pod Template:
   9    Labels:       system=ReplicaOne
   10   Containers:
   11    nginx:
   12     Image:            nginx:1.15.1
   13     Port:             80/TCP
   14     Host Port:        0/TCP
   15     Environment:    <none>
   16     Mounts:         <none>
   17   Volumes:          <none>
   18  Events:             <none>
   ```

5. View the Pods created with the `ReplicaSet`. From the yaml file created there should be two Pods. You may see a `Completed` busybox which will be cleared out eventually.

   `student@master:~$ kubectl get pods`

   ```
   1  NAME            READY      STATUS      RESTARTS    AGE
   2  rs-one-2p9x4    1/1        Running     0           5m4s
   3  rs-one-3c6pb    1/1        Running     0           5m4s
   ```

6. Now we will delete the `ReplicaSet`, but not the Pods it controls.

```
student@master:~$ kubectl delete rs rs-one --cascade=false
```

```
1  replicaset.apps "rs-one" deleted
```

7. View the `ReplicaSet` and Pods again:

```
student@master:~$ kubectl describe rs rs-one
```

```
1  Error from server (NotFound): replicasets.apps "rs-one" not found
```

```
student@master:~$ kubectl get pods
```

```
1  NAME            READY       STATUS      RESTARTS    AGE
2  rs-one-2p9x4    1/1         Running     0           7m
3  rs-one-3c6pb    1/1         Running     0           7m
```

8. Create the `ReplicaSet` again. As long as we do not change the `selector` field, the new `ReplicaSet` should take ownership. Pod software versions cannot be updated this way.

```
student@master:~$ kubectl create -f rs.yaml
```

```
1  replicaset.apps/rs-one created
```

9. View the age of the `ReplicaSet` and then the Pods within:

```
student@master:~$ kubectl get rs
```

```
1  NAME        DESIRED    CURRENT    READY      AGE
2  rs-one      2          2          2          46s
```

```
student@master:~$ kubectl get pods
```

```
1  NAME            READY       STATUS      RESTARTS    AGE
2  rs-one-2p9x4    1/1         Running     0           8m
3  rs-one-3c6pb    1/1         Running     0           8m
```

10. We will now isolate a Pod from its `ReplicaSet`. Begin by editing the label of a Pod. We will change the `system:` parameter to be `IsolatedPod`.

```
student@master:~$ kubectl edit pod rs-one-3c6pb
```

```
....
  labels:
    system: IsolatedPod    #<-- Change from ReplicaOne
managedFields:
....
```

11. View the number of pods within the `ReplicaSet`. You should see two running.

```
student@master:~$ kubectl get rs
```

```
1  NAME        DESIRED    CURRENT    READY      AGE
2  rs-one      2          2          2          4m
```

12. Now view the pods with the label key of `system`. You should note that there are three, with one being newer than others. The `ReplicaSet` made sure to keep two replicas, replacing the Pod which was isolated.

```
student@master:~$ kubectl get po -L system
```

```
1  NAME            READY      STATUS      RESTARTS    AGE        SYSTEM
2  rs-one-3c6pb    1/1        Running     0           10m        IsolatedPod
3  rs-one-2p9x4    1/1        Running     0           10m        ReplicaOne
4  rs-one-dq5xd    1/1        Running     0           30s        ReplicaOne
```

13. Delete the `ReplicaSet`, then view any remaining Pods.

    student@master:~$ kubectl delete rs rs-one

    ```
    1  replicaset.apps "rs-one" deleted
    ```

    student@master:~$ kubectl get po

    ```
    1  NAME            READY      STATUS      RESTARTS    AGE
    2  rs-one-3c6pb    1/1        Running     0           14m
    3  rs-one-dq5xd    0/1        Terminating 0           4m
    ```

14. In the above example the Pods had not finished termination. Wait for a bit and check again. There should be no
    `ReplicaSets`, but one Pod.

    student@master:~$ kubectl get rs

    ```
    1  No resources found in default namespaces.
    ```

    student@master:~$ kubectl get pod

    ```
    1  NAME            READY      STATUS      RESTARTS    AGE
    2  rs-one-3c6pb    1/1        Running     0           16m
    3
    ```

15. Delete the remaining Pod using the label.

    student@master:~$ kubectl delete pod -l system=IsolatedPod

    ```
    1  pod "rs-one-3c6pb" deleted
    ```

✎ Exercise 7.2: Working with DaemonSets

A `DaemonSet` is a watch loop object like a `Deployment` which we have been working with in the rest of the labs. The `DaemonSet`
ensures that when a node is added to a cluster a pods will be created on that node. A `Deployment` would only ensure a
particular number of pods are created in general, several could be on a single node. Using a `DaemonSet` can be helpful to
ensure applications are on each node, helpful for things like metrics and logging especially in large clusters where hardware
may be swapped out often. Should a node be removed from a cluster the `DaemonSet` would ensure the Pods are garbage
collected before removal. Starting with Kubernetes v1.12 the scheduler handles `DaemonSet` deployment which means we can
now configure certain nodes to not have a particular `DaemonSet` pods.

This extra step of automation can be useful for using with products like **ceph** where storage is often added or removed, but
perhaps among a subset of hardware. They allow for complex deployments when used with declared resources like memory,
CPU or volumes.

1. We begin by creating a yaml file. In this case the `kind` would be set to `DaemonSet`. For ease of use we will copy the
   previously created `rs.yaml` file and make a couple edits. Remove the `Replicas: 2` line.

    student@master:~$ cp rs.yaml ds.yaml

    student@master:~$ vim ds.yaml

```
YA
ML     ds.yaml
1  ....
2  kind: DaemonSet
3  ....
4    name: ds-one
5  ....
6    replicas: 2 #<<<----Remove this line
7  ....
8          system: DaemonSetOne #<<-- Edit both references
9  ....
```

2. Create and verify the newly formed DaemonSet. There should be one Pod per node in the cluster.

   student@master:~$ kubectl create -f ds.yaml

```
1  daemonset.apps/ds-one created
```

   student@master:~$ kubectl get ds

```
1  NAME        DESIRED    CURRENT    READY       UP-TO-DATE    AVAILABLE    NODE-SELECTOR    AGE
2  ds-one      2          2          2           2             2            <none>           1m
```

   student@master:~$ kubectl get pod

```
1  NAME                  READY       STATUS      RESTARTS    AGE
2  ds-one-b1dcv          1/1         Running     0           2m
3  ds-one-z31r4          1/1         Running     0           2m
```

3. Verify the image running inside the Pods. We will use this information in the next section.

   student@master:~$ kubectl describe pod ds-one-b1dcv | grep Image:

```
1      Image:                nginx:1.15.1
```

✎ Exercise 7.3: Rolling Updates and Rollbacks

One of the advantages of micro-services is the ability to replace and upgrade a container while continuing to respond to client requests. We will use the `OnDelete` setting that upgrades a container when the predecessor is deleted, then the use the `RollingUpdate` feature as well, which begins a rolling update immediately.

⚠️ **nginx versions**

The **nginx** software updates on a distinct timeline from Kubernetes. If the lab shows an older version please use the current default, and then a newer version. Versions can be seen with this command: **sudo docker image ls nginx**

1. Begin by viewing the current `updateStrategy` setting for the `DaemonSet` created in the previous section.

   student@master:~$ kubectl get ds ds-one -o yaml \
        | grep -A 3 Strategy

```
   updateStrategy:
     rollingUpdate:
       maxUnavailable: 1
     type: RollingUpdate
```

2. Edit the object to use the `OnDelete` update strategy. This would allow the manual termination of some of the pods, resulting in an updated image when they are recreated.

   ```
   student@master:~$ kubectl edit ds ds-one
   ```

   ```
   ....
     updateStrategy:
       rollingUpdate:
         maxUnavailable: 1
       type: OnDelete                #<-- Edit to be this line
   status:
   ....
   ```

3. Update the `DaemonSet` to use a newer version of the **nginx** server. This time use the **set** command instead of **edit**. Set the version to be `1.16.1-alpine`.

   ```
   student@master:~$ kubectl set image ds ds-one nginx=nginx:1.16.1-alpine
   ```

   ```
   daemonset.apps/ds-one image updated
   ```

4. Verify that the `Image:` parameter for the Pod checked in the previous section is unchanged.

   ```
   student@master:~$ kubectl describe po ds-one-b1dcv |grep Image:
   ```

   ```
       Image:                 nginx:1.15.1
   ```

5. Delete the Pod. Wait until the replacement Pod is running and check the version.

   ```
   student@master:~$ kubectl delete po ds-one-b1dcv
   ```

   ```
   pod "ds-one-b1dcv" deleted
   ```

   ```
   student@master:~$ kubectl get pod
   ```

   ```
   NAME                   READY      STATUS      RESTARTS    AGE
   ds-one-xc86w           1/1        Running     0           19s
   ds-one-z31r4           1/1        Running     0           4m8s
   ```

   ```
   student@master:~$ kubectl describe pod ds-one-xc86w |grep Image:
   ```

   ```
       Image:                 nginx:1.16.1-alpine
   ```

6. View the image running on the older Pod. It should still show version 1.15.1.

   ```
   student@master:~$ kubectl describe pod ds-one-z31r4 |grep Image:
   ```

   ```
       Image:                 nginx:1.15.1
   ```

7. View the history of changes for the `DaemonSet`. You should see two revisions listed. As we did not use the `--record` option we didn't see why the object updated.

   ```
   student@master:~$ kubectl rollout history ds ds-one
   ```

   ```
   daemonsets "ds-one"
   REVISION          CHANGE-CAUSE
   1                    <none>
   2                    <none>
   ```

8. View the settings for the various versions of the `DaemonSet`. The `Image:` line should be the only difference between the two outputs.

```
student@master:~$ kubectl rollout history ds ds-one --revision=1
```

```
1  daemonsets "ds-one" with revision #1
2  Pod Template:
3    Labels:         system=DaemonSetOne
4    Containers:
5     nginx:
6      Image:          nginx:1.15.1
7      Port:          80/TCP
8      Environment:       <none>
9      Mounts:         <none>
10   Volumes:         <none>
```

```
student@master:~$ kubectl rollout history ds ds-one --revision=2
```

```
1  ....
2     Image:         nginx:1.16.1-alpine
3  .....
```

9. Use `kubectl rollout undo` to change the `DaemonSet` back to an earlier version. As we are still using the `OnDelete` strategy there should be no change to the Pods.

```
student@master:~$ kubectl rollout undo ds ds-one --to-revision=1
```

```
1  daemonset.apps/ds-one rolled back
```

```
student@master:~$ kubectl describe pod ds-one-xc86w |grep Image:
```

```
1      Image:            nginx:1.16.1-alpine
```

10. Delete the Pod, wait for the replacement to spawn then check the image version again.

```
student@master:~$ kubectl delete pod ds-one-xc86w
```

```
1  pod "ds-one-xc86w" deleted
```

```
student@master:~$ kubectl get pod
```

```
1  NAME                READY   STATUS        RESTARTS   AGE
2  ds-one-qc72k        1/1     Running       0          10s
3  ds-one-xc86w        0/1     Terminating   0          12m
4  ds-one-z31r4        1/1     Running       0          28m
```

```
student@master:~$ kubectl describe po ds-one-qc72k |grep Image:
```

```
1      Image:            nginx:1.15.1
```

11. View the details of the `DaemonSet`. The Image should be v1.15.1 in the output.

```
student@master:~$ kubectl describe ds |grep Image:
```

```
1      Image:            nginx:1.15.1
```

12. View the current configuration for the `DaemonSet` in YAML output. Look for the `updateStrategy:` the the `type:`

```
student@master:~$ kubectl get ds ds-one -o yaml
```

```
apiVersion: apps/v1
kind: DaemonSet
.....
      terminationGracePeriodSeconds: 30
  updateStrategy:
    type: OnDelete
status:
  currentNumberScheduled: 2
.....
```

13. Create a new `DaemonSet`, this time setting the update policy to `RollingUpdate`. Begin by generating a new config file.

    ```
    student@master:~$ kubectl get ds ds-one -o yaml  > ds2.yaml
    ```

14. Edit the file. Change the name, around line 69 and the update strategy around line 100, back to the default `RollingUpdate`.

    ```
    student@master:~$ vim ds2.yaml
    ```

    ```
    ....
      name: ds-two
    ....
        type: RollingUpdate
    ```

15. Create the new `DaemonSet` and verify the **nginx** version in the new pods.

    ```
    student@master:~$ kubectl create -f ds2.yaml
    ```

    ```
    daemonset.apps/ds-two created
    ```

    ```
    student@master:~$ kubectl get pod
    ```

    ```
    NAME                 READY    STATUS    RESTARTS   AGE
    ds-one-qc72k         1/1      Running   0          28m
    ds-one-z31r4         1/1      Running   0          57m
    ds-two-10khc         1/1      Running   0          5m
    ds-two-kzp9g         1/1      Running   0          5m
    ```

    ```
    student@master:~$ kubectl describe po ds-two-10khc |grep Image:
    ```

    ```
        Image:                nginx:1.15.1
    ```

16. Edit the configuration file and set the image to a newer version such as 1.16.1-alpine. Include the `--record` option.

    ```
    student@master:~$ kubectl edit ds ds-two --record
    ```

    ```
    ....
        - image: nginx:1.16.1-alpine
    .....
    ```

17. View the age of the `DaemonSets`. It should be around ten minutes old, depending on how fast you type.

    ```
    student@master:~$ kubectl get ds ds-two
    ```

    ```
    NAME      DESIRED   CURRENT   READY     UP-TO-DATE   AVAILABLE   NODE-SELECTOR   AGE
    ds-two    2         2         2         2            2           <none>          10m
    ```

18. Now view the age of the Pods. Two should be much younger than the `DaemonSet`. They are also a few seconds apart due to the nature of the rolling update where one then the other pod was terminated and recreated.

```
student@master:~$ kubectl get pod
```

```
1  NAME                 READY     STATUS     RESTARTS   AGE
2  ds-one-qc72k         1/1       Running    0          36m
3  ds-one-z31r4         1/1       Running    0          1h
4  ds-two-2p8vz         1/1       Running    0          34s
5  ds-two-8lx7k         1/1       Running    0          32s
```

19. Verify the Pods are using the new version of the software.

```
student@master:~$ kubectl describe po ds-two-8lx7k |grep Image:
```

```
1      Image:               nginx:1.16.1-alpine
```

20. View the rollout status and the history of the `DaemonSets`.

```
student@master:~$ kubectl rollout status ds ds-two
```

```
1  daemon set "ds-two" successfully rolled out
```

```
student@master:~$ kubectl rollout history ds ds-two
```

```
1  daemonsets "ds-two"
2  REVISION          CHANGE-CAUSE
3  1                  <none>
4  2                   kubectl edit ds ds-two --record=true
```

21. View the changes in the update they should look the same as the previous history, but did not require the Pods to be deleted for the update to take place.

```
student@master:~$ kubectl rollout history ds ds-two --revision=2
```

```
1  ...
2      Image:         nginx:1.16.1-alpine
```

22. Clean up the system by removing the `DaemonSets`.

```
student@master:~$ kubectl delete ds ds-one ds-two
```

```
1  daemonset.apps "ds-one" deleted
2  daemonset.apps "ds-two" deleted
```

Chapter 8

Volumes and Data

8.1 Labs

✏ Exercise 8.1: Create a ConfigMap

> **Overview**
>
> Container files are ephemeral, which can be problematic for some applications. Should a container be restarted the files will be lost. In addition, we need a method to share files between containers inside a Pod.
>
> A `Volume` is a directory accessible to containers in a Pod. Cloud providers offer volumes which persist further than the life of the Pod, such that AWS or GCE volumes could be pre-populated and offered to Pods, or transferred from one Pod to another. **Ceph** is also another popular solution for dynamic, persistent volumes.
>
> Unlike current **Docker** volumes a Kubernetes volume has the lifetime of the Pod, not the containers within. You can also use different types of volumes in the same Pod simultaneously, but Volumes cannot mount in a nested fashion. Each must have their own mount point. Volumes are declared with `spec.volumes` and mount points with `spec.containers.volumeMounts` parameters. Each particular volume type, 24 currently, may have other restrictions. https://kubernetes.io/docs/concepts/storage/volumes/#types-of-volumes
>
> We will also work with a `ConfigMap`, which is basically a set of key-value pairs. This data can be made available so that a Pod can read the data as environment variables or configuration data. A `ConfigMap` is similar to a `Secret`, except they are not base64 byte encoded arrays. They are stored as strings and can be read in serialized form.

There are three different ways a `ConfigMap` can ingest data, from a literal value, from a file or from a directory of files.

1. We will create a `ConfigMap` containing primary colors. We will create a series of files to ingest into the `ConfigMap`. First, we create a directory `primary` and populate it with four files. Then we create a file in our home directory with our favorite color.

   ```
   student@master:~$ mkdir primary

   student@master:~$ echo c > primary/cyan

   student@master:~$ echo m > primary/magenta
   ```

```
student@master:~$ echo y > primary/yellow

student@master:~$ echo k > primary/black

student@master:~$ echo "known as key" >> primary/black

student@master:~$ echo blue > favorite
```

2. Now we will create the `ConfigMap` and populate it with the files we created as well as a literal value from the command line.

```
student@master:~$ kubectl create configmap colors \
     --from-literal=text=black  \
     --from-file=./favorite  \
     --from-file=./primary/
```

```
1  configmap/colors created
```

3. View how the data is organized inside the cluster. Use the `yaml` then the `json` output type to see the formatting.

```
student@master:~$ kubectl get configmap colors
```

```
1  NAME        DATA        AGE
2  colors      6           30s
```

```
student@master:~$ kubectl get configmap colors -o yaml
```

```
1   apiVersion: v1
2   data:
3     black: |
4       k
5       known as key
6     cyan: |
7       c
8     favorite: |
9       blue
10    magenta: |
11      m
12    text: black
13    yellow: |
14      y
15  kind: ConfigMap
16  <output_omitted>
```

4. Now we can create a Pod to use the `ConfigMap`. In this case a particular parameter is being defined as an environment variable.

```
student@master:~$ vim simpleshell.yaml
```

simpleshell.yaml

```
1   apiVersion: v1
2   kind: Pod
3   metadata:
4     name: shell-demo
5   spec:
6     containers:
7     - name: nginx
8       image: nginx
```

```
 9        env:
10        - name: ilike
11          valueFrom:
12            configMapKeyRef:
13              name: colors
14              key: favorite
```

5. Create the Pod and view the environmental variable. After you view the parameter, exit out and delete the pod.

    student@master:~$ kubectl create -f simpleshell.yaml

```
1  pod/shell-demo created
```

    student@master:~$ kubectl exec shell-demo -- /bin/bash -c 'echo $ilike'

```
1  blue
```

    student@master:~$ kubectl delete pod shell-demo

```
1  pod "shell-demo" deleted
```

6. All variables from a file can be included as environment variables as well. Comment out the previous `env:` stanza and add a slightly different `envFrom` to the file. Having new and old code at the same time can be helpful to see and understand the differences. Recreate the Pod, check all variables and delete the pod again. They can be found spread throughout the environment variable output.

    student@master:~$ vim simpleshell.yaml

simpleshell.yaml

```
 1  <output_omitted>
 2      image: nginx
 3  #    env:
 4  #    - name: ilike
 5  #      valueFrom:
 6  #        configMapKeyRef:
 7  #          name: colors
 8  #          key: favorite
 9      envFrom:                    #<-- Same indent as image: line
10      - configMapRef:
11          name: colors
```

    student@master:~$ kubectl create -f simpleshell.yaml

```
1  pod/shell-demo created
```

    student@master:~$ kubectl exec shell-demo -- /bin/bash -c 'env'

```
1  black=k
2  known as key
3
4  KUBERNETES_SERVICE_PORT_HTTPS=443
5  cyan=c
6  <output_omitted>
```

                         THE LINUX FOUNDATION

```
student@master:~$ kubectl delete pod shell-demo
```

```
1  pod "shell-demo" deleted
```

7. A `ConfigMap` can also be created from a YAML file. Create one with a few parameters to describe a car.

```
student@master:~$ vim car-map.yaml
```

car-map.yaml

```
1  apiVersion: v1
2  kind: ConfigMap
3  metadata:
4    name: fast-car
5    namespace: default
6  data:
7    car.make: Ford
8    car.model: Mustang
9    car.trim: Shelby
```

8. Create the `ConfigMap` and verify the settings.

```
student@master:~$ kubectl create -f car-map.yaml
```

```
1  configmap/fast-car created
```

```
student@master:~$ kubectl get configmap fast-car -o yaml
```

```
1  apiVersion: v1
2  data:
3    car.make: Ford
4    car.model: Mustang
5    car.trim: Shelby
6  kind: ConfigMap
7  <output_omitted>
```

9. We will now make the `ConfigMap` available to a Pod as a mounted volume. You can again comment out the previous environmental settings and add the following new stanza. The `containers:` and `volumes:` entries are indented the same number of spaces.

```
student@master:~$ vim simpleshell.yaml
```

simpleshell.yaml

```
1  <output_omitted>
2  spec:
3    containers:
4      - name: nginx
5        image: nginx
6        volumeMounts:
7        - name: car-vol
8          mountPath: /etc/cars
9    volumes:
10     - name: car-vol
11       configMap:
```

```
12          name: fast-car
13  <comment out rest of file>
```

10. Create the Pod again. Verify the volume exists and the contents of a file within. Due to the lack of a carriage return in the file your next prompt may be on the same line as the output, `Shelby`.

    student@master:~$ kubectl create -f simpleshell.yaml

    ```
    1  pod "shell-demo" created
    ```

    student@master:~$ kubectl exec shell-demo -- /bin/bash -c 'df -ha |grep car'

    ```
    1  /dev/root        9.6G  3.2G   6.4G  34% /etc/cars
    ```

    student@master:~$ kubectl exec shell-demo -- /bin/bash -c 'cat /etc/cars/car.trim'

    ```
    1  Shelby   #<-- Then your prompt
    ```

11. Delete the `Pod` and `ConfigMaps` we were using.

    student@master:~$ kubectl delete pods shell-demo

    ```
    1  pod "shell-demo" deleted
    ```

    student@master:~$ kubectl delete configmap fast-car colors

    ```
    1  configmap "fast-car" deleted
    2  configmap "colors" deleted
    ```

✎ Exercise 8.2: Creating a Persistent NFS Volume (PV)

We will first deploy an NFS server. Once tested we will create a persistent NFS volume for containers to claim.

1. Install the software on your master node.

    student@master:~$ sudo apt-get update && sudo \
                 apt-get install -y nfs-kernel-server

    ```
    1  <output_omitted>
    ```

2. Make and populate a directory to be shared. Also give it similar permissions to /tmp/

    student@master:~$ sudo mkdir /opt/sfw

    student@master:~$ sudo chmod 1777 /opt/sfw/

    student@master:~$ sudo bash -c 'echo software > /opt/sfw/hello.txt'

3. Edit the NFS server file to share out the newly created directory. In this case we will share the directory with all. You can always **snoop** to see the inbound request in a later step and update the file to be more narrow.

    student@master:~$ sudo vim /etc/exports

    ```
    1  /opt/sfw/ *(rw,sync,no_root_squash,subtree_check)
    ```

4. Cause /etc/exports to be re-read:

```
student@master:~$ sudo exportfs -ra
```

5. Test by mounting the resource from your **second** node.

```
student@worker:~$ sudo apt-get -y install nfs-common
```

```
1  <output_omitted>
```

```
student@worker:~$ showmount -e k8smaster
```

```
1  Export list for k8smaster:
2  /opt/sfw  *
```

```
student@worker:~$ sudo mount k8smaster:/opt/sfw /mnt
```

```
student@worker:~$ ls -l /mnt
```

```
1  total 4
2  -rw-r--r-- 1 root root 9 Sep 28 17:55 hello.txt
```

6. Return to the master node and create a YAML file for the object with `kind`, `PersistentVolume`. Use the hostname of the master server and the directory you created in the previous step. Only syntax is checked, an incorrect name or directory will not generate an error, but a Pod using the resource will not start. Note that the `accessModes` do not currently affect actual access and are typically used as labels instead.

```
student@master:~$ vim PVol.yaml
```

PVol.yaml

```
1  apiVersion: v1
2  kind: PersistentVolume
3  metadata:
4    name: pvvol-1
5  spec:
6    capacity:
7      storage: 1Gi
8    accessModes:
9    - ReadWriteMany
10   persistentVolumeReclaimPolicy: Retain
11   nfs:
12     path: /opt/sfw
13     server: k8smaster   #<-- Edit to match master node
14     readOnly: false
```

7. Create the persistent volume, then verify its creation.

```
student@master:~$ kubectl create -f PVol.yaml
```

```
1  persistentvolume/pvvol-1 created
```

```
student@master:~$ kubectl get pv
```

```
1  NAME      CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS
2     CLAIM      STORAGECLASS   REASON    AGE
3  pvvol-1   1Gi        RWX           Retain          Available   4s
```

✏️ Exercise 8.3: Creating a Persistent Volume Claim (PVC)

Before Pods can take advantage of the new PV we need to create a **Persistent Volume Claim** (**PVC**).

1. Begin by determining if any currently exist.

   ```
   student@master:~$ kubectl get pvc
   ```

   ```
   1  No resources found in default namespace.
   ```

2. Create a YAML file for the new pvc.

   ```
   student@master:~$ vim pvc.yaml
   ```

   YAML **pvc.yaml**

   ```
   1   apiVersion: v1
   2   kind: PersistentVolumeClaim
   3   metadata:
   4     name: pvc-one
   5   spec:
   6     accessModes:
   7     - ReadWriteMany
   8     resources:
   9       requests:
   10        storage: 200Mi
   ```

3. Create and verify the new pvc is bound. Note that the size is 1Gi, even though 200Mi was suggested. Only a volume of at least that size could be used.

   ```
   student@master:~$ kubectl create -f pvc.yaml
   ```

   ```
   1  persistentvolumeclaim/pvc-one created
   ```

   ```
   student@master:~$ kubectl get pvc
   ```

   ```
   1  NAME       STATUS    VOLUME    CAPACITY    ACCESSMODES    STORAGECLASS    AGE
   2  pvc-one    Bound     pvvol-1   1Gi         RWX                            4s
   ```

4. Look at the status of the pv again, to determine if it is in use. It should show a status of Bound.

   ```
   student@master:~$ kubectl get pv
   ```

   ```
   1  NAME        CAPACITY    ACCESSMODES    RECLAIMPOLICY    STATUS    CLAIM
   2     STORAGECLASS    REASON    AGE
   3  pvvol-1    1Gi          RWX                            Retain            Bound     default/pvc-one
   4                                 5m
   ```

5. Create a new deployment to use the pvc. We will copy and edit an existing deployment yaml file. We will change the deployment name then add a volumeMounts section under containers and volumes section to the general spec. The name used must match in both places, whatever name you use. The claimName must match an existing pvc. As shown in the following example. The `volumes` line is the same indent as `containers` and `dnsPolicy`.

   ```
   student@master:~$ cp first.yaml nfs-pod.yaml
   ```

   ```
   student@master:~$ vim nfs-pod.yaml
   ```

YAML `nfs-pod.yaml`

```yaml
 1  apiVersion: apps/v1
 2  kind: Deployment
 3  metadata:
 4    annotations:
 5      deployment.kubernetes.io/revision: "1"
 6    generation: 1
 7    labels:
 8      run: nginx
 9    name: nginx-nfs                    #<-- Edit name
10    namespace: default
11  spec:
12    replicas: 1
13    selector:
14      matchLabels:
15        run: nginx
16    strategy:
17      rollingUpdate:
18        maxSurge: 1
19        maxUnavailable: 1
20      type: RollingUpdate
21    template:
22      metadata:
23        creationTimestamp: null
24        labels:
25          run: nginx
26      spec:
27        containers:
28        - image: nginx
29          imagePullPolicy: Always
30          name: nginx
31          volumeMounts:
32          - name: nfs-vol
33            mountPath: /opt
34          ports:
35          - containerPort: 80
36            protocol: TCP
37          resources: {}
38          terminationMessagePath: /dev/termination-log
39          terminationMessagePolicy: File
40        volumes:                          #<<-- These four lines
41        - name: nfs-vol
42          persistentVolumeClaim:
43            claimName: pvc-one
44        dnsPolicy: ClusterFirst
45        restartPolicy: Always
46        schedulerName: default-scheduler
47        securityContext: {}
48        terminationGracePeriodSeconds: 30
```

6. Create the pod using the newly edited file.

   `student@master:~$ kubectl create -f nfs-pod.yaml`

```
1  deployment.apps/nginx-nfs created
```

7. Look at the details of the pod. You may see the `daemonset` pods running as well.

   `student@master:~$ kubectl get pods`

                     THE LINUX FOUNDATION

```
1  NAME                       READY    STATUS    RESTARTS    AGE
2  nginx-nfs-1054709768-s8g28  1/1      Running   0           3m
```

```
student@master:~$ kubectl describe pod nginx-nfs-1054709768-s8g28
```

```
1  Name:           nginx-nfs-1054709768-s8g28
2  Namespace:      default
3  Priority:       0
4  Node:           worker/10.128.0.5
5
6  <output_omitted>
7
8     Mounts:
9       /opt from nfs-vol (rw)
10
11 <output_omitted>
12
13 Volumes:
14   nfs-vol:
15     Type:          PersistentVolumeClaim (a reference to a PersistentV...
16     ClaimName:        pvc-one
17     ReadOnly:        false
18 <output_omitted>
```

8. View the status of the PVC. It should show as bound.

```
student@master:~$ kubectl get pvc
```

```
1  NAME      STATUS  VOLUME   CAPACITY  ACCESS MODES  STORAGECLASS  AGE
2  pvc-one   Bound   pvvol-1  1Gi       RWX                         2m
```

✎ Exercise 8.4: Using a ResourceQuota to Limit PVC Count and Usage

The flexibility of cloud-based storage often requires limiting consumption among users. We will use the ResourceQuota object to both limit the total consumption as well as the number of persistent volume claims.

1. Begin by deleting the deployment we had created to use NFS, the pv and the pvc.

```
student@master:~$ kubectl delete deploy nginx-nfs
```

```
1  deployment.apps "nginx-nfs" deleted
```

```
student@master:~$ kubectl delete pvc pvc-one
```

```
1  persistentvolumeclaim "pvc-one" deleted
```

```
student@master:~$ kubectl delete pv pvvol-1
```

```
1  persistentvolume "pvvol-1" deleted
```

2. Create a yaml file for the ResourceQuota object. Set the storage limit to ten claims with a total usage of 500Mi.

```
student@master:~$ vim storage-quota.yaml
```

YAML `storage-quota.yaml`

```
1  apiVersion: v1
2  kind: ResourceQuota
3  metadata:
4    name: storagequota
5  spec:
6    hard:
7      persistentvolumeclaims: "10"
8      requests.storage: "500Mi"
```

3. Create a new namespace called `small`. View the namespace information prior to the new quota. Either the long name with double dashes `--namespace` or the nickname `ns` work for the resource.

   `student@master:~$ kubectl create namespace small`

   ```
   1  namespace/small created
   ```

   `student@master:~$ kubectl describe ns small`

   ```
   1  Name:          small
   2  Labels:        <none>
   3  Annotations:   <none>
   4  Status:        Active
   5
   6  No resource quota.
   7
   8  No resource limits.
   ```

4. Create a new pv and pvc in the small namespace.

   `student@master:~$ kubectl -n small create -f PVol.yaml`

   ```
   1  persistentvolume/pvvol-1 created
   ```

   `student@master:~$ kubectl -n small create -f pvc.yaml`

   ```
   1  persistentvolumeclaim/pvc-one created
   ```

5. Create the new resource quota, placing this object into the `small` namespace.

   `student@master:~$ kubectl -n small create -f storage-quota.yaml`

   ```
   1  resourcequota/storagequota created
   ```

6. Verify the `small` namespace has quotas. Compare the output to the same command above.

   `student@master:~$ kubectl describe ns small`

   ```
   1   Name:          small
   2   Labels:        <none>
   3   Annotations:   <none>
   4   Status:        Active
   5
   6   Resource Quotas
   7    Name:                    storagequota
   8    Resource                 Used   Hard
   9    --------                 ---    ---
   10   persistentvolumeclaims   1      10
   11   requests.storage         200Mi  500Mi
   12
   13  No resource limits.
   ```

7. Remove the namespace line from the `nfs-pod.yaml` file. Should be around line 11 or so. This will allow us to pass other namespaces on the command line.

   student@master:~$ vim nfs-pod.yaml

8. Create the container again.

   student@master:~$ kubectl -n small create -f nfs-pod.yaml

   ```
   1  deployment.apps/nginx-nfs created
   ```

9. Determine if the deployment has a running pod.

   student@master:~$ kubectl -n small get deploy

   ```
   1  NAME        READY  UP-TO-DATE  AVAILABLE   AGE
   2  nginx-nfs   1/1    1           1           43s
   ```

   student@master:~$ kubectl -n small describe deploy nginx-nfs

   ```
   1  <output_omitted>
   ```

10. Look to see if the pods are ready.

    student@master:~$ kubectl -n small get pod

    ```
    1  NAME                        READY    STATUS    RESTARTS   AGE
    2  nginx-nfs-2854978848-g3khf  1/1      Running   0          37s
    ```

11. Ensure the Pod is running and is using the NFS mounted volume. If you pass the namespace first `Tab` will auto-complete the pod name.

    student@master:~$ kubectl -n small describe pod \
              nginx-nfs-2854978848-g3khf

    ```
    1  Name:          nginx-nfs-2854978848-g3khf
    2  Namespace:     small
    3  <output_omitted>
    4
    5     Mounts:
    6        /opt from nfs-vol (rw)
    7  <output_omitted>
    ```

12. View the quota usage of the namespace

    student@master:~$ kubectl describe ns small

    ```
    1  <output_omitted>
    2
    3  Resource Quotas
    4   Name:                  storagequota
    5   Resource               Used    Hard
    6   --------               ---     ---
    7   persistentvolumeclaims 1       10
    8   requests.storage       200Mi   500Mi
    9
    10 No resource limits.
    ```

13. Create a 300M file inside of the `/opt/sfw` directory on the host and view the quota usage again. Note that with NFS the size of the share is not counted against the deployment.

```
student@master:~$ sudo dd if=/dev/zero of=/opt/sfw/bigfile bs=1M count=300
```

```
1  300+0 records in
2  300+0 records out
3  314572800 bytes (315 MB, 300 MiB) copied, 0.196794 s, 1.6 GB/s
```

```
student@master:~$ kubectl describe ns small
```

```
1  <output_omitted>
2  Resource Quotas
3   Name:                        storagequota
4   Resource                Used         Hard
5   --------                ---          ---
6   persistentvolumeclaims      1          10
7   requests.storage      200Mi         500Mi
8  <output_omitted>
```

```
student@master:~$ du -h /opt/
```

```
1  301M        /opt/sfw
2  41M         /opt/cni/bin
3  41M         /opt/cni
4  341M        /opt/
```

14. Now let us illustrate what happens when a deployment requests more than the quota. Begin by shutting down the existing deployment.

```
student@master:~$ kubectl -n small get deploy
```

```
1  NAME        READY   UP-TO-DATE   AVAILABLE    AGE
2  nginx-nfs   1       1            1            11m
```

```
student@master:~$ kubectl -n small delete deploy nginx-nfs
```

```
1  deployment.apps "nginx-nfs" deleted
```

15. Once the Pod has shut down view the resource usage of the namespace again. Note the storage did not get cleaned up when the pod was shut down.

```
student@master:~$ kubectl describe ns small
```

```
1  <output_omitted>
2  Resource Quotas
3   Name:                   storagequota
4   Resource            Used     Hard
5   --------            ---      ---
6   persistentvolumeclaims 1      10
7   requests.storage      200Mi   500Mi
```

16. Remove the pvc then view the pv it was using. Note the RECLAIM POLICY and STATUS.

```
student@master:~$ kubectl -n small get pvc
```

```
1  NAME       STATUS     VOLUME     CAPACITY    ACCESSMODES    STORAGECLASS    AGE
2  pvc-one    Bound      pvvol-1    1Gi         RWX                            19m
```

```
student@master:~$ kubectl -n small delete pvc pvc-one
```

```
1  persistentvolumeclaim "pvc-one" deleted
```

```
student@master:~$ kubectl -n small get pv
```

```
1 NAME        CAPACITY    ACCESSMODES    RECLAIMPOLICY    STATUS      CLAIM
2 STORAGECLASS    REASON      AGE
3 pvvol-1  1Gi    RWX     Retain   Released    small/pvc-one   44m
```

17. Dynamically provisioned storage uses the `ReclaimPolicy` of the `StorageClass` which could be `Delete`, `Retain`, or some types allow `Recycle`. Manually created persistent volumes default to `Retain` unless set otherwise at creation. The default storage policy is to retain the storage to allow recovery of any data. To change this begin by viewing the yaml output.

```
student@master:~$ kubectl get pv/pvvol-1 -o yaml
```

```
1 ....
2     path: /opt/sfw
3     server: k8smaster
4   persistentVolumeReclaimPolicy: Retain
5 status:
6   phase: Released
```

18. Currently we will need to delete and re-create the object. Future development on a deleter plugin is planned. We will re-create the volume and allow it to use the `Retain` policy, then change it once running.

```
student@master:~$ kubectl delete pv/pvvol-1
```

```
1 persistentvolume "pvvol-1" deleted
```

```
student@master:~$ grep Retain PVol.yaml
```

```
1   persistentVolumeReclaimPolicy: Retain
```

```
student@master:~$ kubectl create -f PVol.yaml
```

```
1 persistentvolume "pvvol-1" created
```

19. We will use `kubectl patch` to change the retention policy to `Delete`. The yaml output from before can be helpful in getting the correct syntax.

```
student@master:~$  kubectl patch pv pvvol-1 -p \
'{"spec":{"persistentVolumeReclaimPolicy":"Delete"}}'
```

```
1 persistentvolume/pvvol-1 patched
```

```
student@master:~$ kubectl get pv/pvvol-1
```

```
1 NAME        CAPACITY    ACCESSMODES    RECLAIMPOLICY    STATUS      CLAIM
2  STORAGECLASS    REASON      AGE
3 pvvol-1   1Gi        RWX           Delete        Available   2m
```

20. View the current quota settings.

```
student@master:~$ kubectl describe ns small
```

```
1 ....
2  requests.storage         0          500Mi
```

21. Create the pvc again. Even with no pods running, note the resource usage.

```
student@master:~$ kubectl -n small create -f pvc.yaml
```

```
1  persistentvolumeclaim/pvc-one created
```

```
student@master:~$ kubectl describe ns small
```

```
1  ....
2   requests.storage          200Mi          500Mi
```

22. Remove the existing quota from the namespace.

```
student@master:~$ kubectl -n small get resourcequota
```

```
1  NAME              CREATED AT
2  storagequota      2019-11-25T04:10:02Z
```

```
student@master:~$ kubectl -n small delete resourcequota storagequota
```

```
1  resourcequota "storagequota" deleted
```

23. Edit the storagequota.yaml file and lower the capacity to 100Mi.

```
student@master:~$ vim storage-quota.yaml
```

```
1   ....
2       requests.storage: "100Mi"
```

24. Create and verify the new storage quota. Note the hard limit has already been exceeded.

```
student@master:~$ kubectl -n small create -f storage-quota.yaml
```

```
1  resourcequota/storagequota created
```

```
student@master:~$ kubectl describe ns small
```

```
1  ....
2   persistentvolumeclaims          1          10
3   requests.storage              200Mi          100Mi
4
5  No resource limits.
```

25. Create the deployment again. View the deployment. Note there are no errors seen.

```
student@master:~$ kubectl -n small create -f nfs-pod.yaml
```

```
1  deployment.apps/nginx-nfs created
```

```
student@master:~$ kubectl -n small describe deploy/nginx-nfs
```

```
1  Name:                    nginx-nfs
2  Namespace:               small
3  <output_omitted>
```

26. Examine the pods to see if they are actually running.

```
student@master:~$ kubectl -n small get po
```

```
1  NAME                      READY     STATUS     RESTARTS    AGE
2  nginx-nfs-2854978848-vb6bh    1/1       Running    0           58s
```

27. As we were able to deploy more pods even with apparent hard quota set, let us test to see if the reclaim of storage takes place. Remove the deployment and the persistent volume claim.

    `student@master:~$ kubectl -n small delete deploy nginx-nfs`

```
1  deployment.apps "nginx-nfs" deleted
```

    `student@master:~$ kubectl -n small delete pvc/pvc-one`

```
1  persistentvolumeclaim "pvc-one" deleted
```

28. View if the persistent volume exists. You will see it attempted a removal, but failed. If you look closer you will find the error has to do with the lack of a `deleter volume plugin` for NFS. Other storage protocols have a plugin.

    `student@master:~$ kubectl -n small get pv`

```
1  NAME        CAPACITY    ACCESSMODES    RECLAIMPOLICY    STATUS     CLAIM
2        STORAGECLASS    REASON      AGE
3  pvvol-1    1Gi         RWX      Delete    Failed    small/pvc-one    20m
```

29. Ensure the deployment, pvc and pv are all removed.

    `student@master:~$ kubectl delete pv/pvvol-1`

```
1  persistentvolume "pvvol-1" deleted
```

30. Edit the persistent volume YAML file and change the `persistentVolumeReclaimPolicy:` to Recycle.

    `student@master:~$ vim PVol.yaml`

    PVol.yaml

```
1  ....
2     persistentVolumeReclaimPolicy: Recycle
3  ....
```

31. Add a `LimitRange` to the namespace and attempt to create the persistent volume and persistent volume claim again. We can use the `LimitRange` we used earlier.

    `student@master:~$ kubectl -n small create -f low-resource-range.yaml`

```
1  limitrange/low-resource-range created
```

32. View the settings for the namespace. Both quotas and resource limits should be seen.

    `student@master:~$ kubectl describe ns small`

```
1  <output_omitted>
2  Resource Limits
3   Type        Resource    Min  Max  Default Request  Default Limit  ...
4   ----        --------    ---  ---  ---------------  -------------  -...
5   Container   cpu         -    -    500m             1              -
6   Container   memory      -    -    100Mi            500Mi          -
```

33. Create the persistent volume again. View the resource. Note the `Reclaim Policy` is `Recycle`.

    ```
    student@master:~$ kubectl -n small create -f PVol.yaml
    ```

    ```
    1  persistentvolume/pvvol-1 created
    ```

    ```
    student@master:~$ kubectl get pv
    ```

    ```
    1  NAME        CAPACITY    ACCESS MODES    RECLAIM POLICY    STATUS        ...
    2  pvvol-1     1Gi         RWX             Recycle           Available     ...
    ```

34. Attempt to create the persistent volume claim again. The quota only takes effect if there is also a resource limit in effect.

    ```
    student@master:~$ kubectl -n small create -f pvc.yaml
    ```

    ```
    1  Error from server (Forbidden): error when creating "pvc.yaml":
    2   persistentvolumeclaims "pvc-one" is forbidden: exceeded quota:
    3  storagequota, requested: requests.storage=200Mi, used:
    4  requests.storage=0, limited: requests.storage=100Mi
    ```

35. Edit the `resourcequota` to increase the `requests.storage` to 500mi.

    ```
    student@master:~$ kubectl -n small edit resourcequota
    ```

    ```
    ....
    2  spec:
    3    hard:
    4      persistentvolumeclaims: "10"
    5      requests.storage: 500Mi
    6  status:
    7    hard:
    8      persistentvolumeclaims: "10"
    9  ....
    ```

36. Create the `pvc` again. It should work this time. Then create the `deployment` again.

    ```
    student@master:~$ kubectl -n small create -f pvc.yaml
    ```

    ```
    1  persistentvolumeclaim/pvc-one created
    ```

    ```
    student@master:~$ kubectl -n small create -f nfs-pod.yaml
    ```

    ```
    1  deployment.apps/nginx-nfs created
    ```

37. View the namespace settings.

    ```
    student@master:~$ kubectl describe ns small
    ```

    ```
    1  <output_omitted>
    ```

38. Delete the `deployment`. View the status of the `pv` and `pvc`.

    ```
    student@master:~$ kubectl -n small delete deploy nginx-nfs
    ```

    ```
    1  deployment.apps "nginx-nfs" deleted
    ```

    ```
    student@master:~$ kubectl -n small get pvc
    ```

```
1 NAME        STATUS    VOLUME    CAPACITY   ACCESS MODES   STORAGECLASS   AGE
2 pvc-one     Bound     pvvol-1   1Gi        RWX                           7m
```

```
student@master:~$ kubectl -n small get pv
```

```
1 NAME       CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS  CLAIM          ...
2 pvvol-1 1Gi          RWX           Recycle         Bound   small/pvc-one ...
```

39. Delete the `pvc` and check the status of the `pv`. It should show as `Available`.

```
student@master:~$ kubectl -n small delete pvc pvc-one
```

```
1 persistentvolumeclaim "pvc-one" deleted
```

```
student@master:~$ kubectl -n small get pv
```

```
1 NAME     CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM STORA...
2 pvvol-1 1Gi       RWX           Recycle        Available       ...
```

40. Remove the `pv` and any other resources created during this lab.

```
student@master:~$ kubectl delete pv pvvol-1
```

```
1 persistentvolume "pvvol-1" deleted
```

Chapter 9

Services

9.1 Labs

✎ Exercise 9.1: Deploy A New Service

> **Overview**
>
> **Services** (also called **microservices**) are objects which declare a policy to access a logical set of Pods. They are typically assigned with `labels` to allow persistent access to a resource, when front or back end containers are terminated and replaced.
>
> Native applications can use the `Endpoints` API for access. Non-native applications can use a Virtual IP-based bridge to access back end pods. `ServiceTypes Type` could be:
>
> - **ClusterIP** default - exposes on a cluster-internal IP. Only reachable within cluster
>
> - **NodePort** Exposes node IP at a static port. A ClusterIP is also automatically created.
>
> - **LoadBalancer** Exposes service externally using cloud providers load balancer. `NodePort` and `ClusterIP` automatically created.
>
> - **ExternalName** Maps service to contents of `externalName` using a `CNAME` record.
>
> We use services as part of decoupling such that any agent or object can be replaced without interruption to access from client to back end application.

1. Deploy two **nginx** servers using **kubectl** and a new `.yaml` file. The kind should be `Deployment` and label it with `nginx`. Create two replicas and expose port 8080. What follows is a well documented file. There is no need to include the comments when you create the file. This file can also be found among the other examples in the tarball.

   ```
   student@master:~$ vim nginx-one.yaml
   ```

```yaml
YAML        nginx-one.yaml

 1  apiVersion: apps/v1
 2  # Determines YAML versioned schema.
 3  kind: Deployment
 4  # Describes the resource defined in this file.
 5  metadata:
 6    name: nginx-one
 7    labels:
 8      system: secondary
 9  # Required string which defines object within namespace.
10    namespace: accounting
11  # Existing namespace resource will be deployed into.
12  spec:
13    selector:
14      matchLabels:
15        system: secondary
16  # Declaration of the label for the deployment to manage
17    replicas: 2
18  # How many Pods of following containers to deploy
19    template:
20      metadata:
21        labels:
22          system: secondary
23  # Some string meaningful to users, not cluster. Keys
24  # must be unique for each object. Allows for mapping
25  # to customer needs.
26      spec:
27        containers:
28  # Array of objects describing containerized application with a Pod.
29  # Referenced with shorthand spec.template.spec.containers
30        - image: nginx:1.16.1
31  # The Docker image to deploy
32          imagePullPolicy: Always
33          name: nginx
34  # Unique name for each container, use local or Docker repo image
35          ports:
36          - containerPort: 8080
37            protocol: TCP
38  # Optional resources this container may need to function.
39        nodeSelector:
40          system: secondOne
41  # One method of node affinity.
```

2. View the existing labels on the nodes in the cluster.

   ```
   student@master:~$ kubectl get nodes --show-labels
   ```

   ```
1  <output_omitted>
   ```

3. Run the following command and look for the errors. Assuming there is no typo, you should have gotten an error about about the `accounting` namespace.

   ```
   student@master:~$ kubectl create -f nginx-one.yaml
   ```

   ```
1  Error from server (NotFound): error when creating
2  "nginx-one.yaml": namespaces "accounting" not found
   ```

4. Create the namespace and try to create the deployment again. There should be no errors this time.

   ```
   student@master:~$ kubectl create ns accounting
   ```

```
1  namespace/accounting" created
```

```
student@master:~$ kubectl create -f nginx-one.yaml
```

```
1  deployment.apps/nginx-one created
```

5. View the status of the new pods. Note they do not show a `Running` status.

```
student@master:~$ kubectl -n accounting get pods
```

```
1  NAME                        READY     STATUS    RESTARTS   AGE
2  nginx-one-74dd9d578d-fcpmv  0/1       Pending   0          4m
3  nginx-one-74dd9d578d-r2d67  0/1       Pending   0          4m
```

6. View the node each has been assigned to (or not) and the reason, which shows under events at the end of the output.

```
student@master:~$ kubectl -n accounting describe pod \
      nginx-one-74dd9d578d-fcpmv
```

```
1  Name:           nginx-one-74dd9d578d-fcpmv
2  Namespace:      accounting
3  Node:           <none>
4
5  <output_omitted>
6
7  Events:
8    Type     Reason           Age         From         ....
9    ----     ------           ----        ----
10   Warning  FailedScheduling <unknown>   default-scheduler
11 0/2 nodes are available: 2 node(s) didn't match node selector.
```

7. Label the secondary node. Note the value is case sensitive. Verify the labels.

```
student@master:~$ kubectl label node worker \
      system=secondOne
```

```
1  node/worker labeled
```

```
student@master:~$ kubectl get nodes --show-labels
```

```
1  NAME              STATUS    ROLES     AGE       VERSION    LABELS
2  master    Ready       master    1d1h      v1.19.0   \
3      beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/
4  hostname=master,node-role.kubernetes.io/master=
5  worker       Ready       <none>    1d1h       v1.19.0   \
6      beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/
7  hostname=worker,system=secondOne
```

8. View the pods in the `accounting` namespace. They may still show as `Pending`. Depending on how long it has been since you attempted deployment the system may not have checked for the label. If the Pods show `Pending` after a minute delete one of the pods. They should both show as `Running` after a deletion. A change in state will cause the `Deployment` controller to check the status of both Pods.

```
student@master:~$  kubectl -n accounting get pods
```

```
1  NAME                        READY     STATUS    RESTARTS   AGE
2  nginx-one-74dd9d578d-fcpmv  1/1       Running   0          10m
3  nginx-one-74dd9d578d-sts5l  1/1       Running   0          3s
```

9. View Pods by the label we set in the YAML file. If you look back the Pods were given a label of `app=nginx`.

```
student@master:~$ kubectl get pods -l system=secondary --all-namespaces
```

```
1  NAMESPACE    NAME                            READY   STATUS    RESTARTS   AGE
2  accounting   nginx-one-74dd9d578d-fcpmv      1/1     Running   0          20m
3  accounting   nginx-one-74dd9d578d-sts5l      1/1     Running   0          9m
```

10. Recall that we exposed port 8080 in the YAML file. Expose the new deployment.

```
student@master:~$ kubectl -n accounting expose deployment nginx-one
```

```
1  service/nginx-one exposed
```

11. View the newly exposed endpoints. Note that port 8080 has been exposed on each Pod.

```
student@lfs458-node-9q6r:~$ kubectl -n accounting get ep nginx-one
```

```
1  NAME          ENDPOINTS                            AGE
2  nginx-one     192.168.1.72:8080,192.168.1.73:8080   47s
```

12. Attempt to access the Pod on port 8080, then on port 80. Even though we exposed port 8080 of the container the application within has not been configured to listen on this port. The **nginx** server listens on port 80 by default. A `curl` command to that port should return the typical welcome page.

```
student@master:~$ curl 192.168.1.72:8080
```

```
1  curl: (7) Failed to connect to 192.168.1.72 port 8080: Connection refused
```

```
student@master:~$ curl 192.168.1.72:80
```

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>Welcome to nginx!</title>
5  <output_omitted>
```

13. Delete the deployment. Edit the YAML file to expose port 80 and create the deployment again.

```
student@master:~$ kubectl -n accounting delete deploy nginx-one
```

```
1  deployment.apps "nginx-one" deleted
```

```
student@master:~$ vim nginx-one.yaml
```

nginx-one.yaml

```
1  ....
2          ports:
3          - containerPort: 8080      #<-- Edit this line
4            protocol: TCP
5  ....
```

```
student@master:~$ kubectl create -f nginx-one.yaml
```

```
1  deployment.apps/nginx-one created
```

✎ Exercise 9.2: Configure a NodePort

In a previous exercise we deployed a `LoadBalancer` which deployed a `ClusterIP` and `NodePort` automatically. In this exercise we will deploy a `NodePort`. While you can access a container from within the cluster, one can use a `NodePort` to NAT traffic from outside the cluster. One reason to deploy a `NodePort` instead, is that a `LoadBalancer` is also a load balancer resource from cloud providers like GKE and AWS.

1. In a previous step we were able to view the **nginx** page using the internal Pod IP address. Now expose the deployment using the `--type=NodePort`. We will also give it an easy to remember name and place it in the `accounting` namespace. We could pass the port as well, which could help with opening ports in the firewall.

   ```
   student@master:~$ kubectl -n accounting expose deployment \
           nginx-one --type=NodePort --name=service-lab
   ```

   ```
   1 service/service-lab exposed
   ```

2. View the details of the services in the `accounting` namespace. We are looking for the autogenerated port.

   ```
   student@master:~$ kubectl -n accounting describe services
   ```

   ```
   1 ....
   2 NodePort:                    <unset>  32103/TCP
   3 ....
   ```

3. Locate the exterior facing hostname or IP address of the cluster. The lab assumes use of GCP nodes, which we access via a `FloatingIP`, we will first check the internal only public IP address. Look for the Kubernetes master URL. Whichever way you access check access using both the internal and possible external IP address

   ```
   student@master:~$ kubectl cluster-info
   ```

   ```
   1 Kubernetes master is running at https://k8smaster:6443
   2 KubeDNS is running at https://k8smaster:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
   3 Metrics-server is running at https://k8smaster:6443/api/v1/namespaces/kube-system/services/https:metrics-server:/
   4 To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
   ```

4. Test access to the **nginx** web server using the combination of master URL and `NodePort`.

   ```
   student@master:~$ curl http://k8smaster:32103
   ```

   ```
   1 <!DOCTYPE html>
   2 <html>
   3 <head>
   4 <title>Welcome to nginx!</title>
   ```

5. Using the browser on your local system, use the public IP address you use to SSH into your node and the port. You should still see the **nginx** default page.

✎ Exercise 9.3: Working with CoreDNS

1. We can leverage **CoreDNS** and predictable hostnames instead of IP addresses. A few steps back we created the `service-lab` NodePort in the Accounting namespace. We will create a new pod for testing using `Ubuntu`. The pod name will be named `nettool`.

   ```
   student@master:~$ vim nettool.yaml
   ```

nettool.yaml

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: nettool
5  spec:
6    containers:
7    - name: ubuntu
8      image: ubuntu:latest
9      command: [ "sleep" ]
10     args: [ "infinity" ]
```

2. Create the pod and then log into it.

```
student@master:~$ kubectl create -f nettool.yaml
```

```
student@master:~$ kubectl exec -it ubuntu -- /bin/bash
```

On Container

(a) Add some tools for investigating DNS and the network. The installation will ask you the geographic area and timezone information. Someone in Austin would first answer 2. America, then 37 for Chicago, which would be central time

```
root@ubuntu:/#  apt-get update ; apt-get install curl dnsutils -y
```

(b) Use the **dig** command with no options. You should see root name servers, and then information about the DNS server responding, such as the IP address.

```
root@ubuntu:/# dig
```

```
1  ; <<>> DiG 9.16.1-Ubuntu <<>>
2  ;; global options: +cmd
3  ;; Got answer:
4  ;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 3394
5  ;; flags: qr rd ra; QUERY: 1, ANSWER: 13, AUTHORITY: 0, ADDITIONAL: 1
6
7  <output_omitted>
8
9  ;; Query time: 4 msec
10 ;; SERVER: 10.96.0.10#53(10.96.0.10)
11 ;; WHEN: Thu Aug 27 22:06:18 CDT 2020
12 ;; MSG SIZE  rcvd: 431
```

(c) Also take a look at the /etc/resolv.conf file, which will indicate nameservers and default domains to search if no using a Fully Qualified Distinguished Name (FQDN). From the output we can see the first entry is `default.svc.cluster.local.`.

```
root@ubuntu:/# cat /etc/resolv.conf
```

```
1  nameserver 10.96.0.10
2  search default.svc.cluster.local svc.cluster.local cluster.local
3  c.endless-station-188822.internal google.internal
4  options ndots:5
```

(d) Use the **dig** command to view more information about the DNS server. Us the **-x** argument to get the FQDN using the IP we know. Notice the domain name, which uses `.kube-system.svc.cluster.local.`, to match the pod namespaces instead of `default`. Also note the name, `kube-dns`, is the name of a service not a pod.

```
root@ubuntu:/# dig @10.96.0.10 -x 10.96.0.10
```
```
1  ...
2  ;; QUESTION SECTION:
3  ;10.0.96.10.in-addr.arpa.          IN          PTR
4
5  ;; ANSWER SECTION:
6  10.0.96.10.in-addr.arpa. 30          IN          PTR          kube-dns.kube-system.svc.cluster.local.
7
8  ;; Query time: 0 msec
9  ;; SERVER: 10.96.0.10#53(10.96.0.10)
10 ;; WHEN: Thu Aug 27 23:39:14 CDT 2020
11 ;; MSG SIZE  rcvd: 139
```

(e) Recall the name of the `service-lab` service we made and the namespaces it was created in. Use this information to create a FQDN and view the exposed pod.

```
root@ubuntu:/# curl service-lab.accounting.svc.cluster.local.
```
```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>Welcome to nginx!</title>
5  <style>
6      body {
7          width: 35em;
8          margin: 0 auto;
9          font-family: Tahoma, Verdana, Arial, sans-serif;
10     }
11 ...
```

(f) Attempt to view the default page using just the service name. It should fail as `nettool` is in the default namespace.

```
root@ubuntu:/# curl service-lab
```
```
1  curl: (6) Could not resolve host: service-lab
```

(g) Add the `accounting` namespaces to the name and try again. Traffic can access a service using a name, even across different namespaces.

```
root@ubuntu:/# curl service-lab.accounting
```
```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>Welcome to nginx!</title>
5  <output_omitted>
```

(h) Exit out of the container and look at the services running inside of the `kube-system` namespace. From the output we see that the `kube-dns` service has the DNS serverIP, and exposed ports DNS uses.

```
root@ubuntu:/# exit
```

```
student@master:~$ kubectl -n kube-system get svc
```
```
1  NAME       TYPE        CLUSTER-IP     EXTERNAL-IP    PORT(S)                 AGE
2  kube-dns   ClusterIP   10.96.0.10     <none>         53/UDP,53/TCP,9153/TCP  42h
```

3. Examine the service in detail. Among other information notice the selector in use to determine the pods the service communicates with.

    `student@master:~$ kubectl -n kube-system get svc kube-dns -o yaml`

```
1   ...
2       labels:
3         k8s-app: kube-dns
4         kubernetes.io/cluster-service: "true"
5         kubernetes.io/name: KubeDNS
6   ...
7       selector:
8         k8s-app: kube-dns
9       sessionAffinity: None
10      type: ClusterIP
11  ...
```

4. Find pods with the same labels in all namespaces. We see that infrastructure pods all have this label, including **?**coredns?

    `student@master:~$ kubectl get pod -l k8s-app --all-namespaces`

```
1  NAMESPACE      NAME                                      READY   STATUS    RESTARTS   AGE
2  kube-system    calico-kube-controllers-5447dc9cbf-275fs  1/1     Running   0          41h
3  kube-system    calico-node-6q74j                         1/1     Running   0          43h
4  kube-system    calico-node-vgzg2                         1/1     Running   0          42h
5  kube-system    coredns-f9fd979d6-4dxpl                   1/1     Running   0          41h
6  kube-system    coredns-f9fd979d6-nxfrz                   1/1     Running   0          41h
7  kube-system    kube-proxy-f4vxx                          1/1     Running   0          41h
8  kube-system    kube-proxy-pdxwd                          1/1     Running   0          41h
```

5. Look at the details of one of the `coredns` pods. Read through the pod spec and find the image in use as well as any configuration information. You should find that configuration comes from a configmap.

    `student@master:~$ kubectl -n kube-system get pod coredns-f9fd979d6-4dxpl -o yaml`

```
1   ...
2   spec:
3     containers:
4     - args:
5       - -conf
6       - /etc/coredns/Corefile
7       image: k8s.gcr.io/coredns:1.7.0
8   ...
9       volumeMounts:
10      - mountPath: /etc/coredns
11        name: config-volume
12        readOnly: true
13  ...
14    volumes:
15    - configMap:
16        defaultMode: 420
17        items:
18        - key: Corefile
19          path: Corefile
20        name: coredns
21      name: config-volume
22  ...
```

6. View the configmaps in the `kube-system` namespace.

    `student@master:~$ kubectl -n kube-system get configmaps`

```
1  NAME                                    DATA    AGE
2  calico-config                           4       43h
3  coredns                                 1       43h
4  extension-apiserver-authentication      6       43h
5  kube-proxy                              2       43h
6  kubeadm-config                          2       43h
7  kubelet-config-1.18                     1       43h
8  kubelet-config-1.19                     1       41h
```

7. View the details of the `coredns` configmap. Note the `cluster.local` domain is listed.

   `student@master:~$ kubectl -n kube-system get configmaps coredns -o yaml`

```
1  apiVersion: v1
2  data:
3    Corefile: |
4      .:53 {
5          errors
6          health {
7              lameduck 5s
8          }
9          ready
10         kubernetes cluster.local in-addr.arpa ip6.arpa {
11             pods insecure
12             fallthrough in-addr.arpa ip6.arpa
13             ttl 30
14         }
15         prometheus :9153
16         forward . /etc/resolv.conf {
17             max_concurrent 1000
18         }
19         cache 30
20         loop
21         reload
22         loadbalance
23     }
24 kind: ConfigMap
25 ...
```

8. While there are many options and zone files we could configure, lets start with simple edit. Add a rewrite statement such that `test.io` will redirect to `cluster.local` More about each line can be found at coredns.io.

   `student@master:~$ kubectl -n kube-system edit configmaps coredns`

```
1  apiVersion: v1
2  data:
3    Corefile: |
4      .:53 {
5          rewrite name regex (.*)\.test\.io {1}.default.svc.cluster.local   #<-- Add this line
6          errors
7          health {
8              lameduck 5s
9          }
10         ready
11         kubernetes cluster.local in-addr.arpa ip6.arpa {
12             pods insecure
13             fallthrough in-addr.arpa ip6.arpa
14             ttl 30
15         }
16         prometheus :9153
17         forward . /etc/resolv.conf {
18             max_concurrent 1000
```

```
19          }
20          cache 30
21          loop
22          reload
23          loadbalance
24      }
```

9. Delete the coredns pods causing them to re-read the updated configmap.

student@master:~$ kubectl -n kube-system delete pod coredns-f9fd979d6-s4j98 coredns-f9fd979d6-xlpzf

```
1  pod "coredns-f9fd979d6-s4j98" deleted
2  pod "coredns-f9fd979d6-xlpzf" deleted
```

10. Create a new web server and create a `ClusterIP` service to verify the address works. Note the new service IP to start with a reverse lookup.

student@master:~$ kubectl create deployment nginx --image=nginx

```
1  deployment.apps/nginx created
```

student@master:~$ kubectl expose deployment nginx --type=ClusterIP --port=80

```
1  service/nginx expose
```

student@master:~$ kubectl get svc

```
1  NAME         TYPE        CLUSTER-IP       EXTERNAL-IP    PORT(S)    AGE
2  kubernetes   ClusterIP   10.96.0.1        <none>         443/TCP    3d15h
3  nginx        ClusterIP   10.104.248.141   <none>         80/TCP     7s
```

11. Log into the `ubuntu` container and test the URL rewrite starting with the reverse IP resolution.

student@master:~$ kubectl exec -it ubuntu -- /bin/bash

On Container

(a) Use the **dig** command. Note that the service name becomes part of the FQDN.

root@ubuntu:/# dig -x 10.104.248.141

```
1  ....
2  ;; QUESTION SECTION:
3  ;141.248.104.10.in-addr.arpa.        IN        PTR
4
5  ;; ANSWER SECTION:
6  141.248.104.10.in-addr.arpa. 30        IN        PTR        nginx.default.svc.cluster.local.
7  ....
```

(b) Now that we have the reverse lookup test the forward lookup. The IP should match the one we used in the previous step.

root@ubuntu:/# dig nginx.default.svc.cluster.local.

```
1  ....
2  ;; QUESTION SECTION:
3  ;nginx.default.svc.cluster.local. IN        A
4
5  ;; ANSWER SECTION:
```

```
6  nginx.default.svc.cluster.local. 30 IN        A        10.104.248.141
7  ....
```

(c) Now test to see if the rewrite rule for the `test.io` domain we added resolves the IP. Note the response uses the original name, not the requested FQDN.

`root@ubuntu:/# dig nginx.test.io`

```
1  ....
2  ;; QUESTION SECTION:
3  ;nginx.test.io.                        IN        A
4
5  ;; ANSWER SECTION:
6  nginx.default.svc.cluster.local. 30 IN        A        10.104.248.141
7  ....
```

12. Exit out of the container then edit the configmap to add an `answer` section.

`student@master:~$ kubectl -n kube-system edit configmaps coredns`

```
1  ....
2  data:
3    Corefile: |
4      .:53 {
5          rewrite stop {                                    #<-- Edit this and following two lines
6              name regex (.*)\.test\.io {1}.default.svc.cluster.local
7              answer name (.*)\.default\.svc\.cluster\.local {1}.test.io
8          }
9          errors
10         health {
11
12 ....
```

13. Delete the coredns pods again to ensure they re-read the updated configmap.

`student@master:~$ kubectl -n kube-system delete pod coredns-f9fd979d6-fv9qn coredns-f9fd979d6-lnxn5`

```
1  pod "coredns-f9fd979d6-fv9qn" deleted
2  pod "coredns-f9fd979d6-lnxn5" deleted
```

14. Log into the ubuntu container again. This time the response should show the FQDN with the requested FQDN.

`student@master:~$ kubectl exec -it ubuntu -- /bin/bash`

On Container

`root@ubuntu:/# dig nginx.test.io`

```
1  ....
2  ;; QUESTION SECTION:
3  ;nginx.test.io.                        IN        A
4
5  ;; ANSWER SECTION:
6  nginx.test.io.              30        IN        A        10.104.248.141
7  ....
```

15. Exit then delete the DNS test tools container to recover the resources.

```
student@master:~$ kubectl delete -f nettool.yaml
```

✏️Exercise 9.4: Use Labels to Manage Resources

1. Try to delete all Pods with the `system=secondary` label, in all namespaces.

   ```
   student@master:~$ kubectl delete pods -l system=secondary \
               --all-namespaces
   ```

   ```
   1  pod "nginx-one-74dd9d578d-fcpmv" deleted
   2  pod "nginx-one-74dd9d578d-sts5l" deleted
   ```

2. View the Pods again. New versions of the Pods should be running as the controller responsible for them continues.

   ```
   student@master:~$ kubectl -n accounting get pods
   ```

   ```
   1  NAME                          READY      STATUS     RESTARTS    AGE
   2  nginx-one-74dd9d578d-ddt5r    1/1        Running    0           1m
   3  nginx-one-74dd9d578d-hfzml    1/1        Running    0           1m
   ```

3. We also gave a label to the deployment. View the deployment in the accounting namespace.

   ```
   student@master:~$ kubectl -n accounting get deploy --show-labels
   ```

   ```
   1  NAME        READY   UP-TO-DATE   AVAILABLE   AGE   LABELS
   2  nginx-one   2/2     2            2           10m   system=secondary
   ```

4. Delete the deployment using its label.

   ```
   student@master:~$ kubectl -n accounting delete deploy -l system=secondary
   ```

   ```
   1  deployment.apps "nginx-one" deleted
   ```

5. Remove the label from the secondary node. Note that the syntax is a minus sign directly after the key you want to remove, or `system` in this case.

   ```
   student@master:~$ kubectl label node worker system-
   ```

   ```
   1  node/worker labeled
   ```

Chapter 10

Ingress

THE **LINUX** FOUNDATION

10.1 Labs

✎ Exercise 10.1: Advanced Service Exposure

Configure an Ingress Controller

With such a fast changing project, it is important to keep track of updates. The main place to find documentation of the current version is https://kubernetes.io/.

1. If you have a large number of services to expose outside of the cluster, or to expose a low-number port on the host node you can deploy an ingress controller or a service mesh. While **nginx** and **GCE** have controllers officially supported by Kubernetes.io, the **Traefik** ingress controller is easier to install. At the moment.

   student@master:~$ kubectl create deployment secondapp --image=nginx

   ```
   1  deployment.apps/secondapp created
   ```

2. Find the labels currently in use by the deployment. We will use them to tie traffic from the ingress controller to the proper service.

   student@master:~$ kubectl get deployments secondapp -o yaml |grep ' labels' -A2

   ```
   1    labels:
   2      app: secondapp
   3    name: secondapp
   4  --
   5        labels:
   6          app: secondapp
   7      spec:
   ```

3. Expose the new server as a NodePort using port 80.

   student@master:~$ kubectl expose deployment secondapp \
       --type=NodePort --port=80

97

```
1  service/secondapp exposed
```

4. As we have RBAC configured we need to make sure the controller will run and be able to work with all necessary ports, endpoints and resources. Create a YAML file to declare a `clusterrole` and a `clusterrolebinding`.

   `student@master:~$ vim ingress.rbac.yaml`

ingress.rbac.yaml

```
1   kind: ClusterRole
2   apiVersion: rbac.authorization.k8s.io/v1
3   metadata:
4     name: traefik-ingress-controller
5   rules:
6     - apiGroups:
7         - ""
8       resources:
9         - services
10        - endpoints
11        - secrets
12      verbs:
13        - get
14        - list
15        - watch
16    - apiGroups:
17        - extensions
18      resources:
19        - ingresses
20      verbs:
21        - get
22        - list
23        - watch
24  ---
25  kind: ClusterRoleBinding
26  apiVersion: rbac.authorization.k8s.io/v1
27  metadata:
28    name: traefik-ingress-controller
29  roleRef:
30    apiGroup: rbac.authorization.k8s.io
31    kind: ClusterRole
32    name: traefik-ingress-controller
33  subjects:
34  - kind: ServiceAccount
35    name: traefik-ingress-controller
36    namespace: kube-system
```

5. Create the new role and binding.

   `student@master:~$ kubectl create -f ingress.rbac.yaml`

```
1  clusterrole.rbac.authorization.k8s.io "traefik-ingress-controller" created
2  clusterrolebinding.rbac.authorization.k8s.io "traefik-ingress-controller" created
```

6. Create the Traefik controller. We will use YAML file from the tarball, which is a slightly older version of what you would find at https://traefik.io/. They recently updated to v2.0, and have yet to fix their dashboard. The included YAML file won't work with v2.0.

   Use the **find** command to locate the file, then copy to the current directory. Your location may be different.

   `student@master:~$ find . -name  traefik-ds.yaml`

```
1  ./LFS458/SOLUTIONS/s_10/traefik-ds.yaml
```

`student@master:~$ cp ./LFS458/SOLUTIONS/s_10/traefik-ds.yaml .`

7. Then create the ingress controller using **kubectl create**.

`student@master:~$ kubectl create -f traefik-ds.yaml`

```
1  serviceaccount "traefik-ingress-controller" created
2  daemonset.apps "traefik-ingress-controller" created
3  service "traefik-ingress-service" created
```

8. Now that there is a new controller we need to pass some rules, so it knows how to handle requests. Note that the host mentioned is www.example.com, which is probably not your node name. We will pass a false header when testing. Also the service name needs to match the `secondapp` label we found in an earlier step.

`student@master:~$ vim ingress.rule.yaml`

ingress.rule.yaml

```
1   apiVersion: networking.k8s.io/v1beta1
2   kind: Ingress
3   metadata:
4     name: ingress-test
5     annotations:
6       kubernetes.io/ingress.class: traefik
7   spec:
8     rules:
9     - host: www.example.com
10      http:
11        paths:
12        - backend:
13            serviceName: secondapp
14            servicePort: 80
15          path: /
```

9. Now ingest the rule into the cluster.

`student@master:~$ kubectl create -f ingress.rule.yaml`

```
1  ingress.extensions "ingress-test" created
```

10. We should be able to test the internal and external IP addresses, and see the nginx welcome page. The loadbalancer would present the traffic, a **curl** request in this case, to the externally facing interface. Use the hostname or use **ip a** to find the IP address of the interface which would face the loadbalancer. In this example the interface would be `ens4`, and the IP would be `10.128.0.7`. In our case we already have an alias to use, `k8smaster`.

`student@master:~$ ip a`

```
1  1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
2     link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
3     inet 127.0.0.1/8 scope host lo
4        valid_lft forever preferred_lft forever
5     inet6 ::1/128 scope host
6        valid_lft forever preferred_lft forever
7  2: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc mq state UP group default qlen 1000
8     link/ether 42:01:0a:80:00:03 brd ff:ff:ff:ff:ff:ff
9     inet 10.128.0.7/32 brd 10.128.0.3 scope global ens4
10       valid_lft forever preferred_lft forever
11 <output_omitted>
```

```
student@master:~$ curl -H "Host: www.example.com" http://k8smaster/
```

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>Welcome to nginx!</title>
5  <style>
```

```
student@master:~$ curl -H "Host: www.example.com" http://35.193.3.179
```

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>Welcome to nginx!</title>
5  <style>
6  <output_omitted>
```

11. At this point we would keep adding more and more web servers. We'll configure one more, which would then be a process continued as many times as desired.

    Begin by deploying another nginx server. Give it a label and expose port 80.

```
student@master:~$ kubectl create deployment thirdpage --image=nginx
```

```
1  deployment.apps/thirdpage created
```

12. Find the label for the new deployment. Look for the `name:`, which would be `thirdpage` in this example.

```
student@master:~$ kubectl get deployment thirdpage -o yaml |grep -A2 ' labels'

    labels:
      app: thirdpage
   name: thirdpage
--
      labels:
        app: thirdpage
    spec:
```

13. Expose the new server as a `NodePort`.

```
student@master:~$ kubectl expose deployment \
   thirdpage --type=NodePort --port=80
```

```
1  service/thirdpage exposed
```

14. Now we will customize the installation. Run a bash shell inside the new pod. Your pod name will end differently. Install **vim** inside the container then edit the `index.html` file of nginx so that the title of the web page will be `Third Page`.

```
student@master:~$ kubectl exec -it thirdpage-5cf8d67664-zcmfh -- /bin/bash
```

Inside container

```
root@thirdpage-5cf8d67664-zcmfh:/# apt-get update

<output_omitted >

root@thirdpage-5cf8d67664-zcmfh:/#  apt-get install vim -y

<output_omitted >

root@thirdpage-5cf8d67664-zcmfh:/# vim /usr/share/nginx/html/index.html
```

```
1
2  <!DOCTYPE html>
3  <html>
4  <head>
5  <title>Third Page</title>
6  <style>


   root@thirdpage-5cf8d67664-zcmfh:/# exit
```

15. Edit the ingress rules to forward traffic to the `thridpage` service. Use the `serviceName` we found in an earlier step for `thirdpage`. It may be easiest to copy the existing `secondapp` stanza, paste, and edit.

    `student@master:~$ kubectl edit ingress ingress-test`

```
1   <output_omitted>
2     - host: www.example.com
3       http:
4         paths:
5         - backend:
6             serviceName: secondapp
7             servicePort: 80
8           path: /
9           pathType: ImplementationSpecific
10    - host: thirdpage.org                       #<-- Add this and following lines
11      http:
12        paths:
13        - backend:
14            serviceName: thirdpage
15            servicePort: 80
16          path: /
17          pathType: ImplementationSpecific
18
19   status:
20   <output_omitted>
```

16. Test the second hostname using **curl** locally as well as from a remote system.

    `student@master:~$ curl -H "Host: thirdpage.org" http://k8smaster/`

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>Third Page</title>
5  <style>
6  <output_omitted>
```

17. The **Traefik.io** ingress controller also presents a dashboard which allows you to monitor basic traffic. From your local system open a browser and navigate to the public IP of your master node like this ⟨YOURPUBLICIP⟩:8080/dashboard/. The trailing slash makes a difference.

    Follow the `HEALTH` and `PROVIDERS` links at the top, as well as the the node IP links to view traffic when you reference the pages, from inside or outside the node. Mistype the domain names inside the **curl** commands and you can also see `404 error traffic`. Explore as time permits.
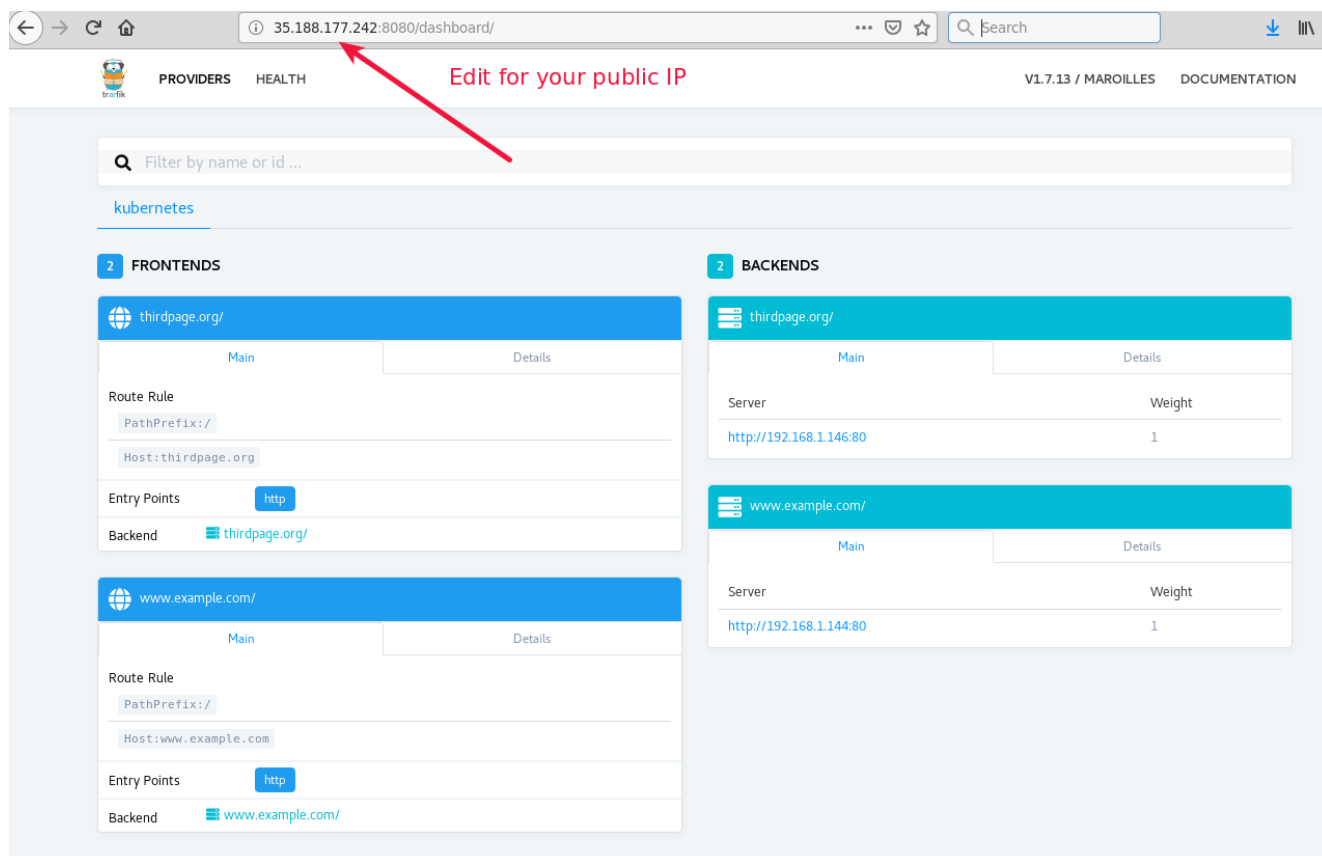
Figure 10.1: **Accessing the API**

Chapter 11

Scheduling

11.1   Labs

✎ Exercise 11.1: Assign Pods Using Labels

> **Overview**
>
> While allowing the system to distribute Pods on your behalf is typically the best route, you may want to determine which nodes a Pod will use. For example you may have particular hardware requirements to meet for the workload. You may want to assign VIP Pods to new, faster hardware and everyone else to older hardware.
>
> In this exercise we will use `labels` to schedule Pods to a particular node. Then we will explore `taints` to have more flexible deployment in a large environment.

1. Begin by getting a list of the nodes. They should be in the ready state and without added labels or taints.

   ```
   student@master:~$ kubectl get nodes
   ```

   ```
   1  NAME            STATUS     ROLES      AGE         VERSION
   2  master    Ready      master     44h         v1.19.0
   3  worker        Ready      <none>    43h          v1.19.0
   ```

2. View the current labels and taints for the nodes.

   ```
   student@master:~$ kubectl describe nodes |grep -A5 -i label
   ```

   ```
   1   Labels:           beta.kubernetes.io/arch=amd64
   2                     beta.kubernetes.io/os=linux
   3                     kubernetes.io/arch=amd64
   4                     kubernetes.io/hostname=c-jg5n
   5                     kubernetes.io/os=linux
   6                     node-role.kubernetes.io/master=
   7   --
   8   Labels:           beta.kubernetes.io/arch=amd64
   9                     beta.kubernetes.io/os=linux
   10                    kubernetes.io/arch=amd64
   ```

```
11                          kubernetes.io/hostname=c-v2t3
12                          kubernetes.io/os=linux
13   --
```

```
student@master:~$ kubectl describe nodes |grep -i taint
```

```
1   Taints:                         <none>
2   Taints:                         <none>
```

3. Get a count of how many containers are running on both the master and worker nodes. There are about 24 containers running on the master in the following example, and eight running on the worker. There are status lines which increase the **wc** count. You may have more or less, depending on previous labs and cleaning up of resources. If you are using **cri-o** you can view containers using **crictl ps**.

```
student@master:~$ kubectl get deployments --all-namespaces
```

```
1   NAMESPACE       NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
2   default         secondapp                 1       1            1           37m
3   default         thirdpage                 1       1            1           24m
4   kube-system     calico-kube-controllers   0       0            0           44h
5   kube-system     coredns                   2/2     2            2           44h
```

```
student@master:~$ sudo docker ps |wc -l
```

```
1   21
```

```
student@worker:~$ sudo docker ps |wc -l
```

```
1   11
```

4. For the purpose of the exercise we will assign the master node to be VIP hardware and the secondary node to be for others.

```
student@master:~$ kubectl label nodes master status=vip
```

```
1   node/master labeled
```

```
student@master:~$ kubectl label nodes worker status=other
```

```
1   node/worker labeled
```

5. Verify your settings. You will also find there are some built in labels such as hostname, os and architecture type. The output below appears on multiple lines for readability.

```
student@master:~$ kubectl get nodes --show-labels
```

```
1   NAME            STATUS    ROLES     AGE     VERSION    LABELS
2   master     Ready     master    44h     v1.19.0    beta.kubernetes.io/arch=
3   amd64,beta.kubernetes.io/os=linux,kubernetes.io/hostname=master,
4   node-role.kubernetes.io/master=,status=vip
5   worker       Ready     <none>    44h     v1.19.0    beta.kubernetes.io/arch=
6   amd64,beta.kubernetes.io/os=linux,kubernetes.io/hostname=worker,status=other
```

6. Create `vip.yaml` to spawn four `busybox` containers which sleep the whole time. Include the `nodeSelector` entry.

```
student@master:~$ vim vip.yaml
```

vip.yaml

```yaml
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: vip
5  spec:
6    containers:
7    - name: vip1
8      image: busybox
9      args:
10     - sleep
11     - "1000000"
12    - name: vip2
13      image: busybox
14      args:
15     - sleep
16     - "1000000"
17    - name: vip3
18      image: busybox
19      args:
20     - sleep
21     - "1000000"
22    - name: vip4
23      image: busybox
24      args:
25     - sleep
26     - "1000000"
27    nodeSelector:
28      status: vip
```

7. Deploy the new pod. Verify the containers have been created on the master node. It may take a few seconds for all the containers to spawn. Check both the master and the secondary nodes.

```
student@master:~$ kubectl create -f vip.yaml
```
```
1  pod/vip created
```

```
student@master:~$ sudo docker ps |wc -l
```
```
1  26
```

```
student@worker:~$ sudo docker ps |wc -l
```
```
1  11
```

8. Delete the pod then edit the file, commenting out the `nodeSelector` lines. It may take a while for the containers to fully terminate.

```
student@master:~$ kubectl delete pod vip
```
```
1  pod "vip" deleted
```

```
student@master:~$ vim vip.yaml
```
```
1  ....
2  #  nodeSelector:
3    #    status: vip
```

9. Create the pod again. Containers should now be spawning on either node. You may see pods for the `daemonsets` as well.

   `student@master:~$ kubectl get pods`

   ```
   1  NAME                          READY    STATUS     RESTARTS    AGE
   2  secondapp-85765cd95c-2q9sx    1/1      Running    0           43m
   3  thirdpage-7c9b56bfdd-2q5pr    1/1      Running    0           30m
   ```

   `student@master:~$ kubectl create -f vip.yaml`

   ```
   1  pod/vip created
   ```

10. Determine where the new containers have been deployed. They should be more evenly spread this time.

    `student@master:~$ sudo docker ps |wc -l`

    ```
    1  21
    ```

    `student@worker:~$ sudo docker ps |wc -l`

    ```
    1  16
    ```

11. Create another file for other users. Change the names from vip to others, and uncomment the nodeSelector lines.

    `student@master:~$ cp vip.yaml other.yaml`

    `student@master:~$ sed -i s/vip/other/g other.yaml`

    `student@master:~$ vim other.yaml`

    other.yaml
    ```
    1  ....
    2     nodeSelector:
    3        status: other
    ```

12. Create the other containers. Determine where they deploy.

    `student@master:~$ kubectl create -f other.yaml`

    ```
    1  pod/other created
    ```

    `student@master:~$ sudo docker ps |wc -l`

    ```
    1  21
    ```

    `student@worker:~$ sudo docker ps |wc -l`

    ```
    1  21
    ```

13. Shut down both pods and verify they terminated. Only our previous pods should be found.

    `student@master:~$ kubectl delete pods vip other`

    ```
    1  pod "vip" deleted
    2  pod "other" deleted
    ```

```
student@master:~$ kubectl get pods
```

```
1  NAME                      READY   STATUS     RESTARTS   AGE
2  secondapp-85765cd95c-2q9sx   1/1   Running    0          51m
3  thirdpage-7c9b56bfdd-2q5pr   1/1   Running    0          40m
```

✐ Exercise 11.2: Using Taints to Control Pod Deployment

> Use taints to manage where Pods are deployed or allowed to run. In addition to assigning a Pod to a group of nodes, you may also want to limit usage on a node or fully evacuate Pods. Using taints is one way to achieve this. You may remember that the master node begins with a `NoSchedule` taint. We will work with three taints to limit or remove running pods.

1. Verify that the master and secondary node have the minimal number of containers running.

   ```
   student@master:~$ kubectl delete deployment secondapp \
         thirdpage
   ```

   ```
   1  deployment.apps "secondapp" deleted
   2  deployment.apps "thirdpage" deleted
   ```

2. Create a deployment which will deploy eight **nginx** containers. Begin by creating a `YAML` file.

   ```
   student@master:~$ vim taint.yaml
   ```

   YAML   **taint.yaml**

   ```yaml
   1  apiVersion: apps/v1
   2  kind: Deployment
   3  metadata:
   4    name: taint-deployment
   5  spec:
   6    replicas: 8
   7    selector:
   8      matchLabels:
   9        app: nginx
   10   template:
   11     metadata:
   12       labels:
   13         app: nginx
   14     spec:
   15       containers:
   16       - name:  nginx
   17         image: nginx:1.16.1
   18         ports:
   19         - containerPort: 80
   ```

3. Apply the file to create the deployment.

   ```
   student@master:~$ kubectl apply -f taint.yaml
   ```

   ```
   1  deployment.apps/taint-deployment created
   ```

4. Determine where the containers are running. In the following example three have been deployed on the master node and five on the secondary node. Remember there will be other housekeeping containers created as well. Your numbers may be slightly different.

```
student@master:~$ sudo docker ps |grep nginx
```
```
1  00c1be5df1e7           nginx@sha256:e3456c851a152494c3e.....
2  <output_omitted>
```

```
student@master:~$ sudo docker ps |wc -l
```
```
1  27
```

```
student@worker:~$ sudo docker ps |wc -l
```
```
1  17
```

5. Delete the deployment. Verify the containers are gone.

```
student@master:~$ kubectl delete deployment taint-deployment
```
```
1  deployment.apps "taint-deployment" deleted
```

```
student@master:~$ sudo docker ps |wc -l
```
```
1  21
```

6. Now we will use a taint to affect the deployment of new containers.   There are three taints, `NoSchedule`, `PreferNoSchedule` and `NoExecute`. The taints having to do with schedules will be used to determine newly deployed containers, but will not affect running containers. The use of `NoExecute` will cause running containers to move.

   Taint the secondary node, verify it has the taint then create the deployment again. We will use the key of `bubba` to illustrate the key name is just some string an admin can use to track Pods.

```
student@master:~$ kubectl taint nodes worker \
          bubba=value:PreferNoSchedule
```
```
1  node/worker tainted
```

```
student@master:~$ kubectl describe node |grep Taint
```
```
1  Taints:                       bubba=value:PreferNoSchedule
2  Taints:                       <none>
```

```
student@master:~$ kubectl apply -f taint.yaml
```
```
1  deployment.apps/taint-deployment created
```

7. Locate where the containers are running. We can see that more containers are on the master, but there still were some created on the secondary. Delete the deployment when you have gathered the numbers.

```
student@master:~$ sudo docker ps |wc -l
```
```
1  21
```

```
student@worker:~$ sudo docker ps |wc -l
```
```
1  23
```

```
student@master:~$ kubectl delete deployment taint-deployment
```
```
1  deployment.apps "taint-deployment" deleted
```

8. Remove the taint, verify it has been removed. Note that the key is used with a minus sign appended to the end.

```
student@master:~$ kubectl taint nodes worker bubba-
```

```
1  node/worker untainted
```

```
student@master:~$ kubectl describe node |grep Taint
```

```
1  Taints:                        <none>
2  Taints:                        <none>
```

9. This time use the `NoSchedule` taint, then create the deployment again. The secondary node should not have any new containers, with only daemonsets and other essential pods running.

```
student@master:~$ kubectl taint nodes worker \
        bubba=value:NoSchedule
```

```
1  node/worker tainted
```

```
student@master:~$ kubectl apply -f taint.yaml
```

```
1  deployment.apps/taint-deployment created
```

```
student@master:~$ sudo docker ps |wc -l
```

```
1  21
```

```
student@worker:~$ sudo docker ps |wc -l
```

```
1  23
```

10. Remove the taint and delete the deployment. When you have determined that all the containers are terminated create the deployment again. Without any taint the containers should be spread across both nodes.

```
student@master:~$ kubectl delete deployment taint-deployment
```

```
1  deployment.apps "taint-deployment" deleted
```

```
student@master:~$ kubectl taint nodes worker bubba-
```

```
1  node/worker untainted
```

```
student@master:~$ kubectl apply -f taint.yaml
```

```
1  deployment.apps/taint-deployment created
```

```
student@master:~$ sudo docker ps |wc -l
```

```
1  27
```

```
student@worker:~$ sudo docker ps |wc -l
```

```
1  17
```

11. Now use the `NoExecute` to taint the secondary (**worker**) node. Wait a minute then determine if the containers have moved. The DNS containers can take a while to shutdown. Some containers will remain on the worker node to continue communication from the cluster.

```
student@master:~$ kubectl taint nodes worker \
        bubba=value:NoExecute
```
```
1 node "worker" tainted
```

```
student@master:~$ sudo docker ps |wc -l
```
```
1 37
```

```
student@worker:~$ sudo docker ps |wc -l
```
```
1 5
```

12. Remove the taint. Wait a minute. Note that all of the containers did not return to their previous placement.

```
student@master:~$ kubectl taint nodes worker bubba-
```
```
1 node/worker untainted
```

```
student@master:~$ sudo docker ps |wc -l
```
```
1 32
```

```
student@worker:~$ sudo docker ps |wc -l
```
```
1 6
```

13. Remove the deployment a final time to free up resources.

```
student@master:~$ kubectl delete deployment taint-deployment
```
```
1 deployment.apps "taint-deployment" deleted
```

Chapter 12

Logging and Troubleshooting

12.1 Labs

✏ Exercise 12.1: Review Log File Locations

> **Overview**
>
> In addition to various logs files and command output, you can use **journalctl** to view logs from the node perspective. We will view common locations of log files, then a command to view container logs. There are other logging options, such as the use of a **sidecar** container dedicated to loading the logs of another container in a pod.
>
> Whole cluster logging is not yet available with Kubernetes. Outside software is typically used, such as **Fluentd**, part of http://fluentd.org/, which is another member project of CNCF.io, like Kubernetes.

Take a quick look at the following log files and web sites. As server processes move from node level to running in containers the logging also moves.

1. If using a **systemd**.based Kubernetes cluster, view the node level logs for **kubelet**, the local Kubernetes agent. Each node will have different contents as this is node specific.

   student@master:~$ journalctl -u kubelet |less

   ```
   1  <output_omitted>
   ```

2. Major Kubernetes processes now run in containers. You can view them from the container or the pod perspective. Use the **find** command to locate the **kube-apiserver** log. Your output will be different, but will be very long. Once you locate the files use the **diff** utility to compare them. There should be no difference, as they are symbolic links to /var/log/pods/. If you follow the links the log files are unique.

   student@master:~$ sudo find / -name "*apiserver*log"

   ```
   1  /var/log/containers/kube-apiserver-u16-12-1-dcb8_kube-system_kube-apiserver-
   2  eddae7079382cd382cd55f8f46b192565dd16b6858206039d49b1ad4693c2a10.log
   3  /var/log/containers/kube-apiserver-u16-12-1-dcb8_kube-system_kube-apiserver-
   4  d00a48877af4ed4c7f8eedf2c7805c77cfabb31fcb453f7d89ffa52fc6ea5f36.log
   ```

111

```
student@master:~$ sudo diff /var/log/containers/kube-apiserver-u16-
12-1-dcb8_kube-system_kube-apiserver-eddae7079382cd382cd55f8f46b192565dd16b68
58206039d49b1ad4693c2a10.log    /var/log/containers/kube-apiserver-u16-12-1-
dcb8_kube-system_kube-apiserver-d00a48877af4ed4c7f8eedf2c7805c77cfabb31fcb453
f7d89ffa52fc6ea5f36.log
```

```
1  <output_omitted>
```

3. Take a look at the log file.

```
student@master:~$ sudo less /var/log/containers/kube-apiserver-u16-
12-1-dcb8_kube-system_kube-apiserver-d00a48877af4ed4c7f8eedf2c7805c77cfabb31f
cb453f7d89ffa52fc6ea5f36.log
```

```
1  <output_omitted>
```

4. Search for and review other log files for `coredns`, `kube-proxy`, and other cluster agents.

5. If **not** on a Kubernetes cluster using **systemd** which collects logs via **journalctl** you can view the text files on the master node.

    (a) `/var/log/kube-apiserver.log`
        Responsible for serving the API

    (b) `/var/log/kube-scheduler.log`
        Responsible for making scheduling decisions

    (c) `/var/log/kube-controller-manager.log`
        Controller that manages replication controllers

6. `/var/log/containers`

   Various container logs

7. `/var/log/pods/`

   More log files for current Pods.

8. Worker Nodes Files (on non-**systemd** systems)

    (a) `/var/log/kubelet.log`
        Responsible for running containers on the node

    (b) `/var/log/kube-proxy.log`
        Responsible for service load balancing

9. More reading: https://kubernetes.io/docs/tasks/debug-application-cluster/debug-service/ and https://kubernetes.io/docs/tasks/debug-application-cluster/determine-reason-pod-failure/

✎ Exercise 12.2: Viewing Logs Output

> Container standard out can be seen via the **kubectl logs** command. If there is no standard out, you would not see any output. In addition, the logs would be destroyed if the container is destroyed.

1. View the current Pods in the cluster. Be sure to view Pods in all namespaces.

```
student@master:~$ kubectl get po --all-namespaces
```

```
1  NAMESPACE      NAME                                       READY      STATUS      RESTARTS    AGE
2  kube-system    calico-kube-controllers-7b9dcdcc5-qg6zd    1/1        Running     0           13m
3  kube-system    calico-node-dr279                          1/1        Running     0           6d1h
4
5  ....
6  kube-system    etcd-master                    1/1         Running     2           44h
7  kube-system    kube-apiserver-master          1/1         Running     2           44h
8  kube-system    kube-controller-manager-master 1/1         Running     2           44h
9  kube-system    kube-scheduler-master          1/1         Running     2           44h
10 ....
```

2. View the logs associated with various infrastructure pods. Using the **Tab** key you can get a list and choose a container. Then you can start typing the name of a pod and use **Tab** to complete the name.

    `student@master:~$ kubectl -n kube-system logs <Tab><Tab>`

```
1   calico-kube-controllers-7b9dcdcc5-qg6zd
2   calico-node-dr279
3   calico-node-xtvfd
4   coredns-5644d7b6d9-k7kts
5   coredns-5644d7b6d9-rnr2v
6   etcd-master
7   kube-apiserver-master
8   kube-controller-manager-master
9   kube-proxy-qhc4f
10  kube-proxy-s56hl
11  kube-scheduler-f-master
12  traefik-ingress-controller-hw5tv
13  traefik-ingress-controller-mcn47
```

    `student@master:~$  kubectl -n kube-system logs \`
    `        kube-apiserver-master`

```
1   Flag --insecure-port has been deprecated, This flag will be removed in a future version.
2   I1119 02:31:14.933023       1 server.go:623] external host was not specified, using 10.128.0.3
3   I1119 02:31:14.933356       1 server.go:149] Version: v1.19.0
4   I1119 02:31:15.595131       1 plugins.go:158] Loaded 11 mutating admission controller(s)
5   successfully in the following order: NamespaceLifecycle,LimitRanger,ServiceAccount,
6   NodeRestriction,TaintNodesByCondition,Priority,DefaultTolerationSeconds,DefaultStorageClass,
7   StorageObjectInUseProtection,MutatingAdmissionWebhook,RuntimeClass.
8   I1119 02:31:15.595357       1 plugins.go:161] Loaded 7 validating admission controller(s)
9   successfully in the following order: LimitRanger,ServiceAccount,Priority,
10  PersistentVolumeClaimResize,ValidatingAdmissionWebhook,RuntimeClass,
11  ResourceQuota.
12  <output_omitted>
```

3. View the logs of other Pods in your cluster.

✎ Exercise 12.3: Adding tools for monitoring and metrics

With the deprecation of **Heapster** the new, integrated **Metrics Server** has been further developed and deployed. The **Prometheus** project of CNCF.io has matured from incubation to graduation, is commonly used for collecting metrics, and should be considered as well.

Configure Metrics

> ⚠️ **Very Important**
>
> The `metrics-server` is written to interact with Docker. If you chose to use crio the logs will show errors and inability to collect metrics.

1. Begin by cloning the software. The **git** command should be installed already. Install it if not found.

   ```
   student@master:~$ git clone \
           https://github.com/kubernetes-incubator/metrics-server.git
   ```
   ```
   1  <output_omitted>
   ```

2. As the software may have changed it is a good idea to read the **README.md** file for updated information.

   ```
   student@master:~$ cd metrics-server/ ; less README.md
   ```
   ```
   1  <output_omitted>
   ```

3. Create the necessary objects. Be aware as new versions are released there may be some changes to the process and the created objects. Use the components.yaml to create the objects. The backslash is not necessary if you type it all on one line.

   ```
   student@master:~$ kubectl create -f \
   https://github.com/kubernetes-sigs/metrics-server/releases/download/v0.3.7/components.yaml
   ```
   ```
   1  clusterrole.rbac.authorization.k8s.io/system:aggregated-metrics-reader created
   2  clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-delegator created
   3  rolebinding.rbac.authorization.k8s.io/metrics-server-auth-reader created
   4  apiservice.apiregistration.k8s.io/v1beta1.metrics.k8s.io created
   5  serviceaccount/metrics-server created
   6  deployment.apps/metrics-server created
   7  service/metrics-server created
   8  clusterrole.rbac.authorization.k8s.io/system:metrics-server created
   9  clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server created
   ```

4. View the current objects, which are created in the `kube-system` namespace. All should show a `Running` status.

   ```
   student@master:~$ kubectl -n kube-system get pods
   ```
   ```
   1  <output_omitted>
   2  kube-proxy-ld2hb                           1/1      Running   0         2d21h
   3  kube-scheduler-u16-1-13-1-2f8c             1/1      Running   0         2d21h
   4  metrics-server-fc6d4999b-b9rjj             1/1      Running   0         42s
   ```

5. Edit the `metrics-server` deployment to allow insecure TLS. The default certificate is x509 self-signed and not trusted by default. In production you may want to configure and replace the certificate. You may encounter other issues as this software is fast-changing. The need for the `kubelet-preferred-address-types` line has been reported on some platforms.

   ```
   student@master:~$ kubectl -n kube-system edit deployment metrics-server
   ```

   ```
   1  ....
   2      spec:
   3        containers:
   4        - args:
   5          - --cert-dir=/tmp
   6          - --secure-port=4443
   ```

```
YA
 A
ML
 7            - --kubelet-insecure-tls                    #<-- Add this line
 8            - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname #<--May be needed
 9         image: k8s.gcr.io/metrics-server/metrics-server:v0.3.7
10 ....
```

6. Test that the metrics server pod is running and does not show errors. At first you should see a few lines showing the container is listening. As the software changes these messages may be slightly different.

   student@master:~$ kubectl -n kube-system logs metrics-server<TAB>

```
1 I0207 14:08:13.383209       1 serving.go:312] Generated self-signed cert
2 (/tmp/apiserver.crt, /tmp/apiserver.key)
3 I0207 14:08:14.078360       1 secure_serving.go:116] Serving securely on
4 [::]:4443
```

7. Test that the metrics working by viewing pod and node metrics. Your output may have different pods. It can take an minute or so for the metrics to populate and not return an error.

   student@master:~$ sleep 120 ; kubectl top pod --all-namespaces

```
1 NAMESPACE       NAME                                        CPU(cores)   MEMORY(bytes)
2 kube-system     calico-kube-controllers-7b9dcdcc5-qg6zd     2m           6Mi
3 kube-system     calico-node-dr279                           23m          22Mi
4 kube-system     calico-node-xtvfd                           21m          22Mi
5 kube-system     coredns-5644d7b6d9-k7kts                    2m           6Mi
6 kube-system     coredns-5644d7b6d9-rnr2v                    3m           6Mi
7 <output_omitted>
```

   student@master:~$ kubectl top nodes

```
1 NAME             CPU(cores)   CPU%    MEMORY(bytes)    MEMORY%
2 master    228m        11%    2357Mi           31%
3 worker       76m         3%    1385Mi           18%
```

8. Using keys we generated in an earlier lab we can also interrogate the API server. Your server IP address will be different.

   student@master:~$ curl --cert ./client.pem \
       --key ./client-key.pem --cacert ./ca.pem \
       https://k8smaster:6443/apis/metrics.k8s.io/v1beta1/nodes

```
{
  "kind": "NodeMetricsList",
  "apiVersion": "metrics.k8s.io/v1beta1",
  "metadata": {
    "selfLink": "/apis/metrics.k8s.io/v1beta1/nodes"
  },
  "items": [
    {
      "metadata": {
        "name": "u16-1-13-1-2f8c",
        "selfLink": "/apis/metrics.k8s.io/v1beta1/nodes/u16-1-13-1-2f8c",
        "creationTimestamp": "2019-01-10T20:27:00Z"
      },
      "timestamp": "2019-01-10T20:26:18Z",
      "window": "30s",
      "usage": {
        "cpu": "215675721n",
        "memory": "2414744Ki"
      }
    },
  <output_omitted>
```

THE LINUX FOUNDATION

Configure the Dashboard

While the dashboard looks nice it has not been a common tool in use. Those that could best develop the tool tend to only use the CLI, so it may lack wanted functionality.

> ### Compatibility With Metric Server
>
> The dashboard has not been updated to work with the **Metrics Server** now that **Heapster** has been deprecated. While there is some interest in getting the metrics to show in the dashboard there has been difficulty finding developers to work on the issue. https://github.com/kubernetes/dashboard/issues/2986

1. Create the dashboard. The short URL in the step below, which has an capital "oh", not number zero, is for this longer URL: https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0-beta6/aio/deploy/recommended.yaml.

   `student@master:~$ kubectl create -f https://bit.ly/2OFQRMy`

   ```
   secret/kubernetes-dashboard-certs created
   serviceaccount/kubernetes-dashboard created
   role.rbac.authorization.k8s.io/kubernetes-dashboard-minimal created
   rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard-minimal created
   deployment.apps/kubernetes-dashboard created
   service/kubernetes-dashboard created
   ```

2. View the current services in all namespaces. Note that the `kubernetes-dashboard` is a `ClusterIP` and part of the `kube-system` namespace.

   `student@master:~$ kubectl get svc --all-namespaces`

   ```
   <output_omitted>
   default                 thirdpage                   NodePort    10.101.190.109   <none>   80:30588/TCP
   kube-system             coredns                     ClusterIP   10.96.0.10       <none>   53/UDP,53/TCP,9153/TCP
   kube-system             metrics-server              ClusterIP   10.109.37.204    <none>   443/TCP
   kube-system             traefik-ingress-service     ClusterIP   10.103.146.169   <none>   80/TCP,8080/TCP
   kubernetes-dashboard    dashboard-metrics-scraper   ClusterIP   10.111.17.237    <none>   8000/TCP
   kubernetes-dashboard    kubernetes-dashboard        ClusterIP   10.105.169.5     <none>   443/TCP

   ```

3. Edit the `kubernetes-dashboard` and change the type to a `NodePort`.

   `student@master:~$ kubectl -n kubernetes-dashboard edit svc kubernetes-dashboard`

   ```
   ....
     selector:
       k8s-app: kubernetes-dashboard
     sessionAffinity: None
     type: NodePort                    #<-- Edit this line
   status:
     loadBalancer: {}
   ```

4. Check the `kubernetes-dashboard` service again. The Type should show as `NodePort`. Take note of the high-numbered port, which is `30968` in the example below. Yours will be different.

   `student@master:~$ kubectl -n kubernetes-dashboard get svc kubernetes-dashboard`

   ```
   NAME                   TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
   kubernetes-dashboard   NodePort   10.107.224.246   <none>        443:30968/TCP  6m39s
   ```

5. There has been some issues with RBAC and the dashboard permissions to see objects. In order to ensure access to view various resources give the dashboard admin access.

```
student@master:~$ kubectl create clusterrolebinding dashaccess \
        --clusterrole=cluster-admin \
        --serviceaccount=kubernetes-dashboard:kubernetes-dashboard
```

```
1  clusterrolebinding.rbac.authorization.k8s.io/dashaccess created
```

6. On your local node open a browser and navigate to an HTTPS URL made of the `Public IP` and the high-numbered port. You will get a message about an insecure connection. Select the **Advanced** button, then **Add Exception...**, then **Confirm Security Exception**. The page should then show the `Kubernetes Dashboard`.

Figure 12.1: **External Access via Browser**

7. We will use the `Token` method to access the dashboard. With RBAC we need to use the proper token, the `kubernetes-dashboard-token` in this case. Find the token, copy it then paste into the login page. The **Tab** key can be helpful to complete the secret name instead of finding the hash.

```
student@master:~$ kubectl -n kubernetes-dashboard describe secrets kubernetes-dashboard-token-<TAB>
```

```
1  ....
2  Data
3  ====
4  ca.crt:     1025 bytes
5  namespace:  11 bytes
6  token:      eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZX
7  JuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY
8  291bnQvc2VjcmV0Lm5hbWUiOiJrdWJlcm5ldGVzLWRhc2hib2FyZC10b2tlbi1wbW04NCIsImt1YmVybmV0ZXMuaW8vc2Vydmlj
9  ZWFjY291bnQvc2VydmljZS1hY2NvdW50Lm5hbWUiOiJrdWJlcm5ldGVzLWRhc2hib2FyZCIsImt1YmVybmV0ZXMuaW8vc2Vydml
10 jZWFjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6IjE5MDY4ZDIzLTE1MTctMTFlOS1hZmMyLTQyMDEwYThlMDAwMyIsInN1Yi
11 I6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDprdWJlLXN5c3RlbTprdWJlcm5ldGVzLWRhc2hib2FyZCJ9.aYTUMWr290pjt5i32rb8
12 qXpq4onn3hLhvz6yLSYexgRd6NYsygVUyqnkRsFE1trg9i1ftNXKJdzkY5kQzN3AcpUTvyj_BvJgzNh3JM9p7QMjI8LHTz4TrRZ
13 rvwJVWitrEn4VnTQuFVcADFD_rKB9FyI_gvT_QiW5fQm24ygTIgfOYd44263oakG8sL64q7UfQNW2wt5SOorMUtybOmX4CXNUYM8
14 G44ejEtv9GW5OsVjEmLIGaoEMX7fctwUN_XCyPdzcCg2WOxRHahBJmbCuLz2SSWL52q4nXQmhTq_L8VDDpt6LjEqXW6LtDJZGjVC
15 s2MnBLerQz-ZAgsVaubbQ
```
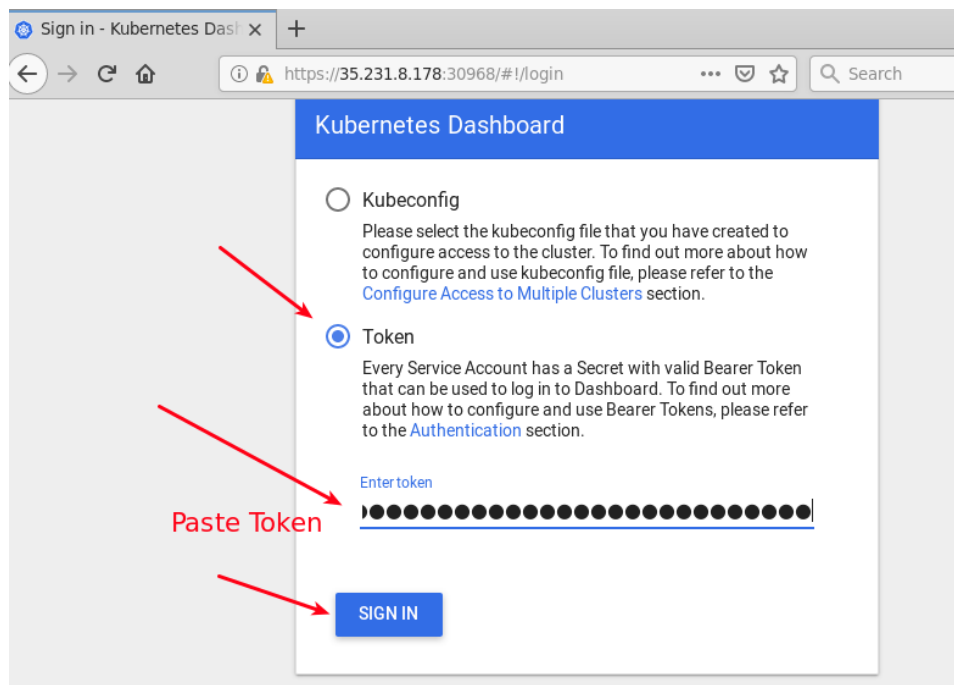
Figure 12.2: **External Access via Browser**

8. Navigate around the various sections and use the menu to the left as time allows. As the pod view is of the `default` namespace, you may want to switch over to the `kube-system` namespace or create a new deployment to view the resources via the GUI. Scale the deployment up and down and watch the responsiveness of the GUI.
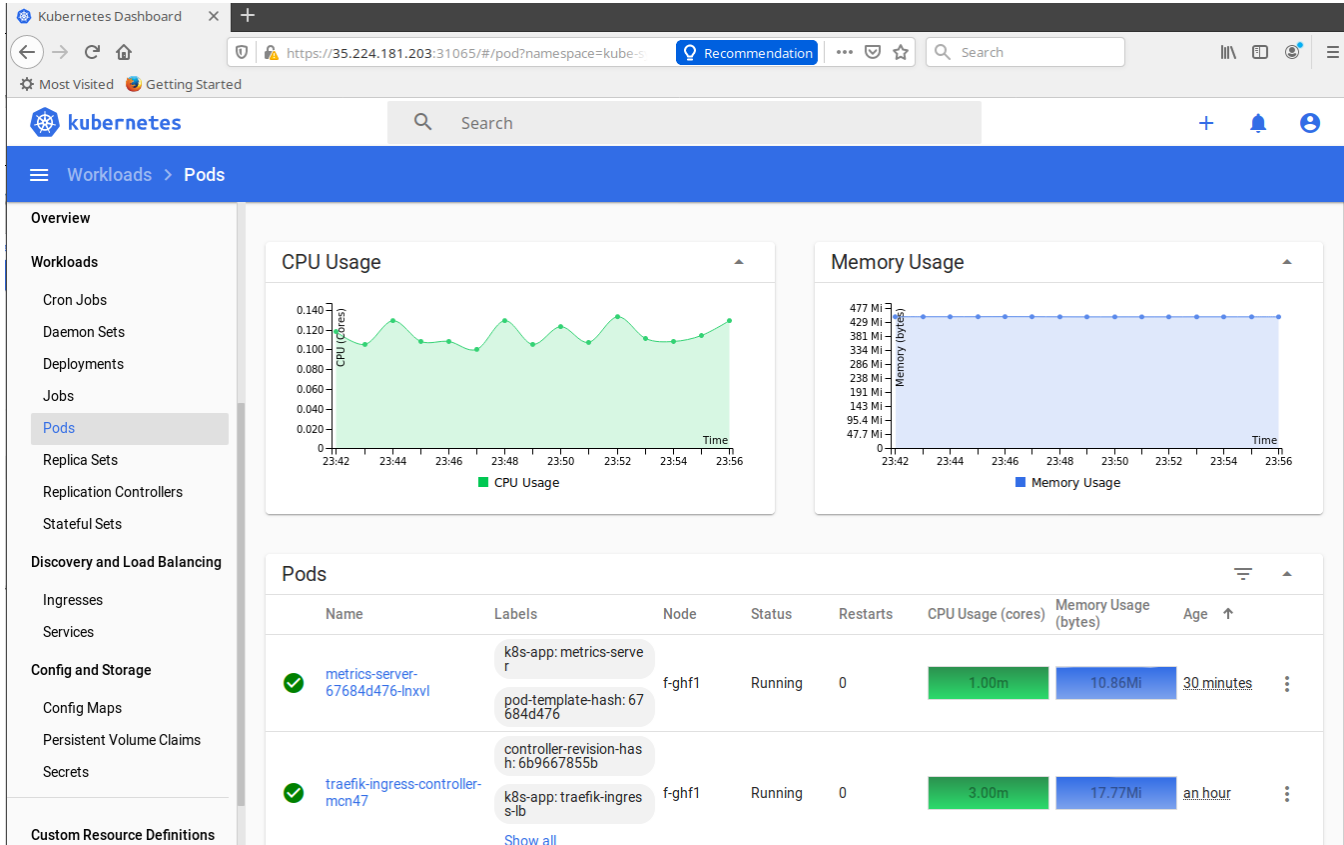
Figure 12.3: **External Access via Browser**

Chapter 13

Custom Resource Definition

13.1 Labs

✏️ Exercise 13.1: Create a Custom Resource Definition

> **Overview**
>
> The use of `CustomResourceDefinitions` (CRD), has become a common manner to deploy new objects and opera-
> tors. Creation of a new operator is beyond the scope of this course, basically it is a watch-loop comparing a spec to
> the current status, and making changes until the states match. A good discussion of creating a controller can be found
> here: https://coreos.com/blog/introducing-operators.html.

First we will examine an existing CRD, then make a simple CRD, but without any particular action. It will be enough to find the
object ingested into the API and responding to commands.

1. View the existing CRDs.

   ```
   student@master:~$ kubectl get crd --all-namespaces
   ```

   ```
   1  NAME                                      CREATED AT
   2  bgpconfigurations.crd.projectcalico.org   2020-04-19T17:29:02Z
   3  bgppeers.crd.projectcalico.org            2020-04-19T17:29:02Z
   4  blockaffinities.crd.projectcalico.org     2020-04-19T17:29:02Z
   5  <output_omitted>
   ```

2. We can see from the names that these CRDs are all working on `Calico`, out network plugin. View the `calico.yaml` file
   we used when we initialized the cluster to see how these objects were created, and some CRD templates to review.

   ```
   student@master:~$ less calico.yaml
   ```

   ```
   1  <output_omitted>
   2  ---
   3  # Source: calico/templates/kdd-crds.yaml
   4
   5  apiVersion: apiextensions.k8s.io/v1beta1
   6  kind: CustomResourceDefinition
   ```

119

```
7  metadata:
8    name: bgpconfigurations.crd.projectcalico.org
9  <output_omitted>
```

3. Now that we have seen some examples, we will create a new YAML file.

   `student@master:~$ vim crd.yaml`

   crd.yaml

   ```
   1  apiVersion: apiextensions.k8s.io/v1beta1
   2  kind: CustomResourceDefinition
   3  metadata:
   4    name: crontabs.training.lfs458.com
   5          # This name must match names below.
   6          # <plural>.<group> syntax
   7  spec:
   8    scope: Cluster    #Could also be Namespaced
   9    group: training.lfs458.com
   10   version: v1
   11   names:
   12     kind: CronTab       #Typically CamelCased for resource manifest
   13     plural: crontabs    #Shown in URL
   14     singular: crontab   #Short name for CLI alias
   15     shortNames:
   16     - ct                #CLI short name
   ```

4. Add the new resource to the cluster.

   `student@master:~$ kubectl create -f crd.yaml`

   ```
   1  customresourcedefinition.apiextensions.k8s.io/crontabs.training.lfs458.com
   2  created
   ```

5. View and describe the resource. The new line may be in the middle of the output. You'll note the **describe** output is unlike other objects we have seen so far.

   `student@master:~$ kubectl get crd`

   ```
   1  NAME                             CREATED AT
   2  <output_omitted>
   3  crontabs.training.lfs458.com     2018-08-03T05:25:20Z
   4  <output_omitted>
   ```

   `student@master:~$ kubectl describe crd crontab<Tab>`

   ```
   1  Name:         crontabs.training.lfs458.com
   2  Namespace:
   3  Labels:       <none>
   4  Annotations:  <none>
   5  API Version:  apiextensions.k8s.io/v1
   6  Kind:         CustomResourceDefinition
   7  <output_omitted>
   ```

6. Now that we have a new API resource we can create a new object of that type. In this case it will be a crontab-like image, which does not actually exist, but is being used for demonstration.

   `student@master:~$ vim new-crontab.yaml`

```yaml
new-crontab.yaml
1   apiVersion: "training.lfs458.com/v1"
2       # This is from the group and version of new CRD
3   kind: CronTab
4       # The kind from the new CRD
5   metadata:
6     name: new-cron-object
7   spec:
8     cronSpec: "*/5 * * * *"
9     image: some-cron-image
10      #Does not exist
```

7. Create the new object and view the resource using short and long name.

   `student@master:~$ kubectl create -f new-crontab.yaml`

   ```
   1  crontab.training.lfs458.com/new-cron-object created
   ```

   `student@master:~$ kubectl get CronTab`

   ```
   1  NAME              AGE
   2  new-cron-object   22s
   ```

   `student@master:~$ kubectl get ct`

   ```
   1  NAME              AGE
   2  new-cron-object   29s
   ```

   `student@master:~$ kubectl describe ct`

   ```
   1  Name:          new-cron-object
   2  Namespace:
   3  Labels:        <none>
   4
   5  <output_omitted>
   6
   7  Spec:
   8    Cron Spec:  */5 * * * *
   9    Image:      some-cron-image
   10 Events:       <none>
   ```

8. To clean up the resources we will delete the CRD. This should delete all of the endpoints and objects using it as well.

   `student@master:~$ kubectl delete -f crd.yaml`

   ```
   1  customresourcedefinition.apiextensions.k8s.io
   2  "crontabs.training.lfs458.com" deleted
   ```

   `student@master:~$ kubectl get ct`

   ```
   1  Error from server (NotFound): Unable to list "training.lfs458.com/v1, Resource=crontabs":
   2  the server could not find the requested resource (get crontabs.training.lfs458.com)
   ```

Chapter 14

Helm

14.1   Labs

✎ Exercise 14.1: Working with Helm and Charts

Overview

helm allows for easy deployment of complex configurations. This could be handy for a vendor to deploy a multi-part application in a single step. Through the use of a `Chart`, or template file, the required components and their relationships are declared. Local agents like **Tiller** use the API to create objects on your behalf. Effectively its orchestration for orchestration.

There are a few ways to install **Helm**. The newest version may require building from source code. We will download a recent, stable version. Once installed we will deploy a `Chart`, which will configure **MariaDB** on our cluster.

Install Helm

1. On the `master` node use **wget** to download the compressed tar file. Various versions can be found here: https://github. com/helm/helm/releases/

   student@master:~$ wget https://get.helm.sh/helm-v3.0.0-linux-amd64.tar.gz

```
1  <output_omitted>
2  helm-v3.0.0-linux-a 100%[===================>]   11.52M  --.-KB/s    in 0.1s
3
4  2019-11-25 06:28:39 (112 MB/s) - 'helm-v3.0.0-linux-amd64.tar.gz' saved [12082866/12082866]
```

2. Uncompress and expand the file.

   student@master:~$ tar -xvf helm-v3.0.0-linux-amd64.tar.gz

```
1  linux-amd64/
2  linux-amd64/helm
3  linux-amd64/README.md
4  linux-amd64/LICENSE
```

3. Copy the **helm** binary to the `/usr/local/bin/` directory, so it is usable via the shell search path.

   ```
   student@master:~$ sudo cp linux-amd64/helm /usr/local/bin/helm3
   ```

4. A `Chart` is a collection of files to deploy an application. There is a good starting repo available on https://github.com/
   kubernetes/charts/tree/master/stable, provided by vendors, or you can make your own. Search the current `Charts` in
   the Helm Hub or an instance of Monocular for available stable databases. Repos change often, so the following output
   may be different from what you see.

   ```
   student@master:~$ helm3 search hub database
   ```

   ```
    1  URL                                            CHART VERSION       APP VERSION
    2           DESCRIPTION
    3  https://hub.helm.sh/charts/choerodon/postgresql 3.18.4             10.7.0
    4           Chart for PostgreSQL, an object-relational data...
    5  https://hub.helm.sh/charts/inspur/redis-cluster 0.0.1             5.0.6
    6           Highly available Kubernetes implementation of R...
    7  https://hub.helm.sh/charts/lohmag/clickhouse    0.2.0             19.17
    8           ClickHouse is an open source column-oriented da...
    9  https://hub.helm.sh/charts/halkeye/turtl        0.1.7             0.7
   10           The secure, collaborative notebook - Totally pr...
   11  <output_omitted>
   ```

5. We will begin by adding a common repository and call it `stable`.

   ```
   student@master:~$ helm3 repo add stable https://kubernetes-charts.storage.googleapis.com
   ```

   ```
    1  "stable" has been added to your repositories
   ```

   ```
   student@master:~$ helm3 repo update
   ```

   ```
    1  Hang tight while we grab the latest from your chart repositories...
    2  ...Successfully got an update from the "stable" chart repository
    3  Update Complete.  Happy Helming!
   ```

6. We will install the **mariadb**. The **- -debug** option will create a lot of output. The output will typically suggest ways to
   access the software. As well we will indicate that we do not want persistent storage, which would require us to create
   an available `PV`.

   ```
   student@master:~$ helm3 --debug install firstdb stable/mariadb \
       --set master.persistence.enabled=false \
       --set slave.persistence.enabled=false
   ```

   ```
    1  install.go:148: [debug] Original chart version: ""
    2  install.go:165: [debug] CHART PATH: /home/student/.cache/helm/repository/mariadb-7.0.1.tgz
    3
    4  client.go:87: [debug] creating 8 resource(s)
    5  NAME: firstdb
    6  LAST DEPLOYED: Mon Nov 25 08:03:30 2019
    7  <output_omitted>
   ```

7. Using some of the information at the end of the previous command output we will deploy another container and access
   the database. We begin by getting the root password for `illmannered-salamander`. Be aware the output lacks a
   carriage return, so the next prompt will appear on the same line. We will need the password to access the running
   `MariaDB` database.

   ```
   student@master:~$ kubectl get secret --namespace default firstdb-mariadb \
       -o jsonpath="{.data.mariadb-root-password}" | base64 --decode
   ```

   ```
    1  TQ93fCLP13
   ```

8. Now we will install another container to act as a client for the database. We will use **apt-get** to install client software. Using copy and paste from the installation output may be helpful.

```
student@master:~$ kubectl run firstdb-mariadb-client \
        --rm --tty -i --restart='Never' \
        --image  docker.io/bitnami/mariadb:10.3.22-debian-10-r27 \
        --namespace default --command -- bash
```

9. Use the client software to access the database. The following command uses the server name and the root password we found in a previous step. Both of yours will be different.

Inside container

```
I have no name!@firstdb-mariadb-client:/$ mysql -h firstdb-mariadb.default.svc.cluster.local \
    -uroot -p my_database
Enter password: TQ93fCLP13
```

```
1  Welcome to the MariaDB monitor.  Commands end with ; or \  g.
2  Your MariaDB connection id is 153
3  Server version: 10.1.38-MariaDB Source distribution
4
5  Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.
6
7  Type 'help;' or '\  h' for help. Type '\  c' to clear the current input statement.
8
9  MariaDB [(none)]> SHOW DATABASES;
10 +--------------------+
11 | Database           |
12 +--------------------+
13 | information_schema |
14 | my_database        |
15 | mysql              |
16 | performance_schema |
17 | test               |
18 +--------------------+
19 5 rows in set (0.00 sec)
20
21 MariaDB [(none)]>
22 MariaDB [(none)]> quit
23 Bye
24 I have no name!@firstdb-mariadb-client:/$ exit
25
```

10. View the `Chart` history on the system. The use of the **-a** option will show all `Charts` including `deleted` and `failed` attempts.

```
student@master:~$ helm3 list
```

```
1  NAME            NAMESPACE       REVISION        UPDATED
2  STATUS          CHART                   APP VERSION
3  firstdb         default         1               2019-11-25 08:03:30.693410615
4  +0000 UTCdeployed       mariadb-7.0.1       10.3.20
```

11. Delete the **mariadb** `Chart`. No releases of mariadb should be found.

```
student@master:~$ helm3 uninstall firstdb
```

```
1  release "firstdb" uninstalled
```

```
student@master:~$ helm3 list
```

```
1 NAME          NAMESPACE        REVISION        UPDATED        STATUS        CHART        APP VERSION
```

12. Find the downloaded chart. It should be a compressed tarball under the user's home directory. Your **mariadb** version may be slightly different.

    ```
    student@master:~$ find $HOME -name *mariadb*
    ```

    ```
    1 /home/student/.cache/helm/repository/mariadb-7.3.14.tgz
    2
    ```

13. Move to the `archive` directory and extract the tarball. Take a look at the files within.

    ```
    student@master:~$ cd $HOME/.cache/helm/repository ; tar -xvf mariadb-*
    ```

    ```
    1 mariadb/Chart.yaml
    2 mariadb/values.yaml
    3 mariadb/templates/NOTES.txt
    4 mariadb/templates/_helpers.tpl
    5 mariadb/templates/initialization-configmap.yaml
    6 <output_omitted>
    ```

14. Copy and rename the `values.yaml` file back to the home directory and return there as well.

    ```
    student@master:~/.cache/helm/repository$ cp mariadb/values.yaml $HOME/custom.yaml ; cd
    ```

15. Review the `custom.yaml` file, note there are many possible configurations. While most are commented out, take a moment to work slowly through the file.

    ```
    student@master:~$ less custom.yaml
    ```

    ```
    1 <output_omitted>
    ```

16. Edit the file. We will change or add two items, the `rootUser password` and storage `persistence` parameters.

    ```
    student@master:~$ vim custom.yaml
    ```

    ```
    1  ....
    2  rootUser:
    3    ## MariaDB admin password
    4    ## ref: https://github.com/bitnami/bitnami-docker-mariadb#setting-the-root-password-on-first-run
    5    ##
    6    password: LFTr@1n                          #<-- Add a password, such as LFTr@1n
    7    ##
    8  ....
    9    persistence:
    10     ## If true, use a Persistent Volume Claim, If false, use emptyDir
    11     ##
    12     enabled: false                           #<--- Change this to false
    13     # Enable persistence using an existing PVC
    14     # existingClaim:
    15 ....
    ```

17. We will now deploy another **MariaDB** instance using our custom settings. Note we no longer have to pass statements to avoid using persistent storage. Near the end of the output will be commands on using the database.

    ```
    student@master:~$ helm3 install -f custom.yaml seconddb stable/mariadb
    ```

```
1  NAME: seconddb
2  LAST DEPLOYED: Mon Nov 25 08:54:08 2019
3  NAMESPACE: default
4  STATUS: deployed
5
6
7  <output_omitted>
8
9  To connect to your database:
10
11    1. Run a pod that you can use as a client:
12
13        kubectl run seconddb-mariadb-client --rm --tty -i --restart='Never' --image
14  docker.io/bitnami/mariadb:10.3.20-debian-9-r0 --namespace default --command -- bash
15
16    2. To connect to master service (read/write):
17
18  <output_omited>
```

18. Copy and paste the command from the output to run a client pod. If you are re-typing, don't include the backlashes. The command could be on typed on a single line. Once in the container log into the database with the password we set and check the default databases.

```
student@master:~$ kubectl run seconddb-mariadb-client \
        --rm --tty -i --restart='Never' \
        --image  docker.io/bitnami/mariadb:10.3.22-debian-10-r27 \
        --namespace default --command -- bash
```

On Container

```
I have no name!@iron-pika-mariadb-client:/$ mysql -h seconddb-mariadb.default.svc.cluster.local \
    -uroot -p my_database
        Enter password: LFTr@1n   #<-- The password we set in the config file
```

```
1  Welcome to the MariaDB monitor.  Commands end with ; or \g.
2  Your MariaDB connection id is 25
3  Server version: 10.3.17-MariaDB-log Source distribution
4
5  Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.
6
7  Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
MariaDB [my_database]> SHOW DATABASES;
```

```
1  +--------------------+
2  | Database           |
3  +--------------------+
4  | information_schema |
5  | my_database        |
6  | mysql              |
7  | performance_schema |
8  | test               |
9  +--------------------+
10 5 rows in set (0.001 sec)
```

```
MariaDB [my_database]> quit
```

```
1  Bye
```

```
I have no name!@seconddb-mariadb-client:/$ exit
```

19. Remove anything you have installed using **helm**. Reference earlier in the chapter if you don't remember the command.

Chapter 15

Security

15.1   Labs

✎ Exercise 15.1: Working with TLS

> **Overview**
>
> We have learned that the flow of access to a cluster begins with TLS connectivity, then authentication followed by authorization, finally an admission control plug-in allows advanced features prior to the request being fulfilled. The use of `Initializers` allows the flexibility of a shell-script to dynamically modify the request. As security is an important, ongoing concern, there may be multiple configurations used depending on the needs of the cluster.
>
> Every process making API requests to the cluster must authenticate or be treated as an anonymous user.

While one can have multiple cluster root Certificate Authorities (CA) by default each cluster uses their own, intended for intra-cluster communication. The CA certificate bundle is distributed to each node and as a secret to default service accounts. The **kubelet** is a local agent which ensures local containers are running and healthy.

1. View the **kubelet** on both the master and secondary nodes. The **kube-apiserver** also shows security information such as certificates and authorization mode. As **kubelet** is a **systemd** service we will start looking at that output.

   student@master:~$ systemctl status kubelet.service

   ```
   1   kubelet.service - kubelet: The Kubernetes Node Agent
   2    Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor preset: en
   3   Drop-In: /etc/systemd/system/kubelet.service.d
   4           |__10-kubeadm.conf
   5  <output_omitted>
   ```

2. Look at the status output. Follow the CGroup and kubelet information, which is a long line where configuration settings are drawn from, to find where the configuration file can be found.

   ```
   1   CGroup: /system.slice/kubelet.service
   2    |--19523 /usr/bin/kubelet .... --config=/var/lib/kubelet/config.yaml ..
   ```

3. Take a look at the settings in the `/var/lib/kubelet/config.yaml` file. Among other information we can see the `/etc/kubernetes/pki/` directory is used for accessing the **kube-apiserver**. Near the end of the output it also sets the directory to find other pod spec files.

   ```
   student@master:~$ sudo less /var/lib/kubelet/config.yaml
   ```

   YAML config.yaml
   ```yaml
    1  address: 0.0.0.0
    2  apiVersion: kubelet.config.k8s.io/v1beta1
    3  authentication:
    4    anonymous:
    5      enabled: false
    6
    7  <output_omitted>
    8
    9  serializeImagePulls: true
   10  staticPodPath: /etc/kubernetes/manifests
   11  streamingConnectionIdleTimeout: 4h0m0s
   12  syncFrequency: 1m0s
   13  topologyManagerPolicy: none
   14  volumeStatsAggPeriod: 1m0s
   ```

4. Other agents on the master node interact with the **kube-apiserver**. View the configuration files where these settings are made. This was set in the previous YAML file. Look at one of the files for cert information.

   ```
   student@master:~$ sudo ls /etc/kubernetes/manifests/
   ```
   ```
   1  etcd.yaml               kube-controller-manager.yaml
   2  kube-apiserver.yaml     kube-scheduler.yaml
   ```

   ```
   student@master:~$ sudo less \
           /etc/kubernetes/manifests/kube-controller-manager.yaml
   ```
   ```
   1  <output_omitted>
   ```

5. The use of `tokens` has become central to authorizing component communication. The tokens are kept as **secrets**. Take a look at the current secrets in the `kube-system` namespace.

   ```
   student@master:~$  kubectl -n kube-system get secrets
   ```
   ```
   1  NAME                                  TYPE
   2    DATA       AGE
   3  attachdetach-controller-token-xqr8n   kubernetes.io/service-account-token
   4    3          5d
   5  bootstrap-signer-token-xbp6s          kubernetes.io/service-account-token
   6    3          5d
   7  bootstrap-token-i3r13t                bootstrap.kubernetes.io/token
   8    7          5d
   9  <output_omitted>
   ```

6. Take a closer look at one of the secrets and the token within. The `certificate-controller-token` could be one to look at. The use of the Tab key can help with long names. Long lines have been truncated in the output below.

   ```
   student@master:~$ kubectl -n kube-system get secrets \
           certificate<Tab> -o yaml
   ```

```
1  apiVersion: v1
2  data:
3    ca.crt: LS0tLS1CRUdJTi.....
4    namespace: a3ViZS1zeXN0ZW0=
5    token: ZX1KaGGJHY2lPaUpTVXpJM....
6  kind: Secret
7  metadata:
8    annotations:
9      kubernetes.io/service-account.name: certificate-controller
10     kubernetes.io/service-account.uid: 7dfa2aa0-9376-11e8-8cfb
11 -42010a800002
12   creationTimestamp: 2018-07-29T21:29:36Z
13   name: certificate-controller-token-wnrwh
14   namespace: kube-system
15   resourceVersion: "196"
16   selfLink: /api/v1/namespaces/kube-system/secrets/certificate-
17 controller-token-wnrwh
18   uid: 7dfbb237-9376-11e8-8cfb-42010a800002
19 type: kubernetes.io/service-account-token
```

7. The **kubectl config** command can also be used to view and update parameters. When making updates this could avoid a typo removing access to the cluster. View the current configuration settings. The keys and certs are redacted from the output automatically.

   student@master:~$ kubectl config view

```
1  apiVersion: v1
2  clusters:
3  - cluster:
4      certificate-authority-data: REDACTED
5  <output_omitted>
```

8. View the options, such as setting a password for the admin instead of a key. Read through the examples and options.

   student@master:~$ kubectl config set-credentials -h

```
1  Sets a user entry in kubeconfig
2  <output_omitted>
```

9. Make a copy of your access configuration file. Later steps will update this file and we can view the differences.

   student@master:~$ cp $HOME/.kube/config $HOME/cluster-api-config

10. Explore working with cluster and security configurations both using **kubectl** and **kubeadm**. Among other values, find the name of your cluster. You will need to become `root` to work with **kubeadm**.

    student@master:~$ kubectl config <Tab><Tab>

```
1  current-context    get-contexts      set-context       view
2  delete-cluster     rename-context    set-credentials
3  delete-context     set               unset
4  get-clusters       set-cluster       use-context
```

   student@master:~$ sudo kubeadm token -h

```
1  <output_omitted>
```

   student@master:~$ sudo kubeadm config -h

```
1  <output_omitted>
```

11. Review the cluster default configuration settings. There may be some interesting tidbits to the security and infrastructure of the cluster.

```
student@master:~$ sudo kubeadm config print init-defaults
```

```
1  apiVersion: kubeadm.k8s.io/v1beta2
2  bootstrapTokens:
3  - groups:
4    - system:bootstrappers:kubeadm:default-node-token
5    token: abcdef.0123456789abcdef
6    ttl: 24h0m0s
7    usages:
8  <output_omitted>
```

✏️Exercise 15.2: Authentication and Authorization

> Kubernetes clusters have two types of users `service accounts` and `normal users`, but `normal users` are assumed to be managed by an outside service. There are no objects to represent them and they cannot be added via an API call, but `service accounts` can be added.
>
> We will use **RBAC** to configure access to actions within a `namespace` for a new contractor, `Developer Dan` who will be working on a new project.

1. Create two namespaces, one for `production` and the other for `development`.

```
student@master:~$ kubectl create ns development
```

```
1  namespace/development created
```

```
student@master:~$ kubectl create ns production
```

```
1  namespace/production created
```

2. View the current clusters and context available. The context allows you to configure the cluster to use, namespace and user for **kubectl** commands in an easy and consistent manner.

```
student@master:~$ kubectl config get-contexts
```

```
1  CURRENT    NAME                       CLUSTER       AUTHINFO          NAMESPACE
2  *          kubernetes-admin@kubernetes   kubernetes    kubernetes-admin
```

3. Create a new user `DevDan` and assign a password of `lfs458`.

```
student@master:~$ sudo useradd -s /bin/bash DevDan
```

```
student@master:~$ sudo passwd DevDan
```

```
1  Enter new UNIX password:  lfs458
2  Retype new UNIX password:  lfs458
3  passwd: password updated successfully
```

4. Generate a private key then Certificate Signing Request (CSR) for `DevDan`. On some Ubuntu 18.04 nodes a missing file may cause an error with random number generation. The **touch** command should ensure one way of success.

```
student@master:~$ openssl genrsa -out DevDan.key 2048
```

```
1  Generating RSA private key, 2048 bit long modulus
2  ......+++
3  .........+++
4  e is 65537 (0x10001)
```

```
student@master:~$ touch $HOME/.rnd

student@master:~$ openssl req -new -key DevDan.key \
     -out DevDan.csr -subj "/CN=DevDan/O=development"
```

5. Using thew newly created request generate a self-signed certificate using the x509 protocol. Use the CA keys for the Kubernetes cluster and set a 45 day expiration. You'll need to use **sudo** to access to the inbound files.

```
student@master:~$ sudo openssl x509 -req -in DevDan.csr \
     -CA /etc/kubernetes/pki/ca.crt \
     -CAkey /etc/kubernetes/pki/ca.key \
     -CAcreateserial \
     -out DevDan.crt -days 45
```

```
1  Signature ok
2  subject=/CN=DevDan/O=development
3  Getting CA Private Key
```

6. Update the access config file to reference the new key and certificate. Normally we would move them to a safe directory instead of a non-root user's home.

```
student@master:~$ kubectl config set-credentials DevDan \
     --client-certificate=/home/student/DevDan.crt \
     --client-key=/home/student/DevDan.key
```

```
1  User "DevDan" set.
```

7. View the update to your credentials file. Use **diff** to compare against the copy we made earlier.

```
student@master:~$ diff cluster-api-config .kube/config
```

```
1  16a,19d15
2  > - name: DevDan
3  >   user:
4  >     as-user-extra: {}
5  >     client-certificate: /home/student/DevDan.crt
6  >     client-key: /home/student/DevDan.key
```

8. We will now create a context. For this we will need the name of the cluster, namespace and CN of the user we set or saw in previous steps.

```
student@master:~$ kubectl config set-context DevDan-context \
     --cluster=kubernetes \
     --namespace=development \
     --user=DevDan
```

```
1  Context "DevDan-context" created.
```

9. Attempt to view the `Pods` inside the `DevDan-context`. Be aware you will get an error.

```
student@master:~$ kubectl --context=DevDan-context get pods
```

```
1  Error from server (Forbidden): pods is forbidden: User "DevDan"
2  cannot list pods in the namespace "development"
```

10. Verify the context has been properly set.

```
student@master:~$ kubectl config get-contexts
```

```
1  CURRENT   NAME                    CLUSTER     AUTHINFO          NAMESPACE
2            DevDan-context          kubernetes  DevDan            development
3  *         kubernetes-admin@kubernetes kubernetes kubernetes-admin
```

11. Again check the recent changes to the cluster access config file.

    `student@master:~$ diff cluster-api-config .kube/config`

    ```
    1  <output_omitted>
    ```

12. We will now create a YAML file to associate RBAC rights to a particular namespace and `Role`.

    `student@master:~$ vim role-dev.yaml`

    role-dev.yaml

    ```
    1  kind: Role
    2  apiVersion: rbac.authorization.k8s.io/v1
    3  metadata:
    4    namespace: development
    5    name: developer
    6  rules:
    7  - apiGroups: ["", "extensions", "apps"]
    8    resources: ["deployments", "replicasets", "pods"]
    9    verbs: ["list", "get", "watch", "create", "update", "patch", "delete"]
    10 # You can use ["*"] for all verbs
    ```

13. Create the object. Check white space and for typos if you encounter errors.

    `student@master:~$ kubectl create -f role-dev.yaml`

    ```
    1  role.rbac.authorization.k8s.io/developer created
    ```

14. Now we create a `RoleBinding` to associate the `Role` we just created with a user. Create the object when the file has been created.

    `student@master:~$ vim rolebind.yaml`

    rolebind.yaml

    ```
    1  kind: RoleBinding
    2  apiVersion: rbac.authorization.k8s.io/v1
    3  metadata:
    4    name: developer-role-binding
    5    namespace: development
    6  subjects:
    7  - kind: User
    8    name: DevDan
    9    apiGroup: ""
    10 roleRef:
    11   kind: Role
    12   name: developer
    13   apiGroup: ""
    ```

    `student@master:~$ kubectl create -f rolebind.yaml`

    ```
    1  rolebinding.rbac.authorization.k8s.io/developer-role-binding created
    ```

15. Test the context again. This time it should work. There are no `Pods` running so you should get a response of `No resources found`.

    `student@master:~$  kubectl --context=DevDan-context get pods`

```
1  No resources found in development namespace.
```

16. Create a new pod, verify it exists, then delete it.

```
student@master:~$ kubectl --context=DevDan-context \
      create deployment  nginx --image=nginx
```

```
1  deployment.apps/nginx created
```

```
student@master:~$ kubectl --context=DevDan-context get pods
```

```
1  NAME                      READY      STATUS          RESTARTS    AGE
2  nginx-7c87f569d-7gb9k     1/1        Running         0           5s
```

```
student@master:~$ kubectl --context=DevDan-context delete \
      deploy nginx
```

```
1  deployment.apps "nginx" deleted
```

17. We will now create a different context for production systems. The `Role` will only have the ability to view, but not create or delete resources. Begin by copying and editing the `Role` and `RoleBindings` YAML files.

```
student@master:~$ cp role-dev.yaml role-prod.yaml
```

```
student@master:~$ vim role-prod.yaml
```

YAML `role-prod.yaml`

```yaml
1  kind: Role
2  apiVersion: rbac.authorization.k8s.io/v1
3  metadata:
4    namespace: production        #<<- This line
5    name: dev-prod              #<<- and this line
6  rules:
7  - apiGroups: ["", "extensions", "apps"]
8    resources: ["deployments", "replicasets", "pods"]
9    verbs: ["get", "list", "watch"] #<<- and this one
```

```
student@master:~$ cp rolebind.yaml rolebindprod.yaml
```

```
student@master:~$ vim rolebindprod.yaml
```

YAML `rolebindprod.yaml`

```yaml
1  kind: RoleBinding
2  apiVersion: rbac.authorization.k8s.io/v1
3  metadata:
4    name: production-role-binding  #<-- Edit to production
5    namespace: production          #<-- Also here
6  subjects:
7  - kind: User
8    name: DevDan
9    apiGroup: ""
10 roleRef:
11   kind: Role
12   name: dev-prod                 #<-- Also this
```

     THE LINUX FOUNDATION

```
YA
ML
13      apiGroup: ""
```

18. Create both new objects.

    ```
    student@master:~$ kubectl create -f role-prod.yaml
    ```

    ```
    1  role.rbac.authorization.k8s.io/dev-prod created
    ```

    ```
    student@master:~$ kubectl create -f rolebindprod.yaml
    ```

    ```
    1  rolebinding.rbac.authorization.k8s.io/production-role-binding created
    ```

19. Create the new context for production use.

    ```
    student@master:~$ kubectl config set-context ProdDan-context \
            --cluster=kubernetes \
            --namespace=production \
            --user=DevDan
    ```

    ```
    1  Context "ProdDan-context" created.
    ```

20. Verify that user DevDan can view pods using the new context.

    ```
    student@master:~$ kubectl --context=ProdDan-context get pods
    ```

    ```
    1  No resources found in production namespace.
    ```

21. Try to create a `Pod` in production. The developer should be `Forbidden`.

    ```
    student@master:~$ kubectl --context=ProdDan-context create \
            deployment nginx --image=nginx
    ```

    ```
    1  Error from server (Forbidden): deployments.apps is forbidden:
    2  User "DevDan" cannot create deployments.apps in the
    3  namespace "production"
    ```

22. View the details of a role.

    ```
    student@master:~$ kubectl -n production describe role dev-prod
    ```

    ```
    1  Name:          dev-prod
    2  Labels:        <none>
    3  Annotations:  kubectl.kubernetes.io/last-applied-configuration=
    4  {"apiVersion":"rbac.authorization.k8s.io/v1","kind":"Role"
    5  ,"metadata":{"annotations":{},"name":"dev-prod","namespace":
    6  "production"},"rules":[{"api...
    7  PolicyRule:
    8    Resources           Non-Resource URLs  Resource Names  Verbs
    9    ---------           -----------------  --------------  -----
    10   deployments         []                 []              [get list watch]
    11   deployments.apps    []                 []              [get list watch]
    12  <output_omitted>
    ```

23. Experiment with other subcommands in both contexts. They should match those listed in the respective roles.

24. **OPTIONAL CHALLENGE STEP:** Become the `DevDan` user. Solve any missing configuration errors. Try to create a deployment in the `development` and the `production` namespaces. Do the errors look the same? Configure as necessary to only have two contexts available to DevDan.

    ```
    DevDan@master:~$ kubectl config get-contexts
    ```

```
1  CURRENT     NAME             CLUSTER      AUTHINFO    NAMESPACE
2  *           DevDan-context   kubernetes   DevDan      development
3              ProdDan-context  kubernetes   DevDan      production
```

✎ Exercise 15.3: Admission Controllers

The last stop before a request is sent to the API server is an `admission control plug-in`. They interact with features such as setting parameters like a default storage class, checking resource quotas, or security settings. A newer feature (v1.7.x) is dynamic controllers which allow new controllers to be ingested or configured at runtime.

1. View the current `admission controller` settings. Unlike earlier versions of Kubernetes the controllers are now compiled into the server, instead of being passed at run-time. Instead of a list of which controllers to use we can enable and disable specific plugins.

   ```
   student@master:~$ sudo grep admission \
       /etc/kubernetes/manifests/kube-apiserver.yaml
   ```

   ```
   1          - --enable-admission-plugins=NodeRestriction
   ```

Chapter 16

High Availability

16.1 Labs

✏ Exercise 16.1: High Availability Steps

> **Overview**
>
> In this lab we will add two more control planes to our cluster, change taints and deploy an application to a particular node, and test that we can access it from outside the cluster. The nodes will handle various infrastructure services and the **etcd** database and should be sized accordingly.
>
> The steps are presented in two ways. First the general steps for those interested in more of a challenge. Following that will be the detailed steps found in previous labs.

You will need three more nodes. One to act as a load balancer, the other two will act as master nodes for quorum. Log into each and use the **ip** command to fill in the table with the IP addresses of the primary interface of each node. If using **GCE** nodes it would be `ens4`, yours may be different. You may need to install software such an editor on the nodes.

Proxy Node	
Second Master	
Third Master	

As the prompts may look similar you may want to change the terminal color or other characteristics to make it easier to keep them distinct. You can also change the prompt using something like: **PS1="ha-proxy$ "**, which may help to keep the terminals distinct.

1. Deploy a load balancer configured to pass through traffic. HAProxy is easy to deploy. Start with just the working master.

2. Install the Kubernetes software on the second and third master.

3. Join the second master to the cluster using the node name. You will need an additional hash and flag from the **kubeadm join** you used to add a worker node.

4. Join the third master to the cluster using the node name.

5. Update the proxy to use all three masters

6. Temporarily shut down the first master and monitor traffic

✎ Exercise 16.2: Detailed Steps

Deploy a Load Balancer

While there are many options, both software and hardware, we will be using an open source tool **HAProxy** to configure a load balancer.

1. Deploy HAProxy. Log into the proxy node. Update the repos then install a the HAProxy software. Answer `yes`, should you the installation ask if you will allow services to restart.

```
student@ha-proxy:~$ sudo apt-get update ; sudo apt-get install -y haproxy vim
```

```
1  <output_omitted>
```

2. Edit the configuration file and add sections for the front-end and back-end servers. We will comment out the second and third master node until we are sure the proxy is forwarding traffic to the known working master.

```
student@ha-proxy:~$ sudo vim /etc/haproxy/haproxy.cfg


....
defaults
        log global                      #<-- Edit these three lines, starting around line 23
        option tcplog
        mode tcp
....
        errorfile 503 /etc/haproxy/errors/503.http
        errorfile 504 /etc/haproxy/errors/504.http

frontend proxynode                      #<-- Add the following lines to bottom of file
    bind *:80
    bind *:6443
    stats uri /proxystats
    default_backend k8sServers

backend k8sServers
    balance roundrobin
    server master  10.128.0.24:6443 check  #<-- Edit these with your IP addresses, port, and hostname
#   server lfs458-SecondMaster  10.128.0.30:6443 check  #<-- Comment out until ready
#   server lfs458-ThirdMaster  10.128.0.66:6443 check   #<-- Comment out until ready
listen stats
    bind :9999
    mode http
    stats enable
    stats hide-version
    stats uri /stats
```

3. Restart the `haproxy` service and check the status. You should see the frontend and backend proxies report being started.

```
student@ha-proxy:~$ sudo systemctl restart haproxy.service
student@ha-proxy:~$ sudo systemctl status haproxy.service
```

```
1  <output_omitted>
2  Aug 08 18:43:08 ha-proxy systemd[1]: Starting HAProxy Load Balancer...
3  Aug 08 18:43:08 ha-proxy systemd[1]: Started HAProxy Load Balancer.
4  Aug 08 18:43:08 ha-proxy haproxy-systemd-wrapper[13602]: haproxy-systemd-wrapper:
5  Aug 08 18:43:08 ha-proxy haproxy[13603]: Proxy proxynode started.
6  Aug 08 18:43:08 ha-proxy haproxy[13603]: Proxy proxynode started.
7  Aug 08 18:43:08 ha-proxy haproxy[13603]: Proxy k8sServers started.
8  Aug 08 18:43:08 ha-proxy haproxy[13603]: Proxy k8sServers started.
```

4. **On the master** Edit the `/etc/hosts` file and comment out the old and add a new `k8smaster` alias to the IP address of the proxy server.

   ```
   student@master:~$ sudo vim /etc/hosts
   ```

   ```
   10.128.0.64 k8smaster        #<-- Add alias to proxy IP
   #10.128.0.24 k8smaster       #<-- Comment out the old alias, in case its needed
   127.0.0.1 localhost
   ....
   ```

5. Use a local browser to navigate to the public IP of your proxy server. The http://34.69.XX.YY:9999/stats is an example your IP address would be different. Leave the browser up and refresh as you run following steps.
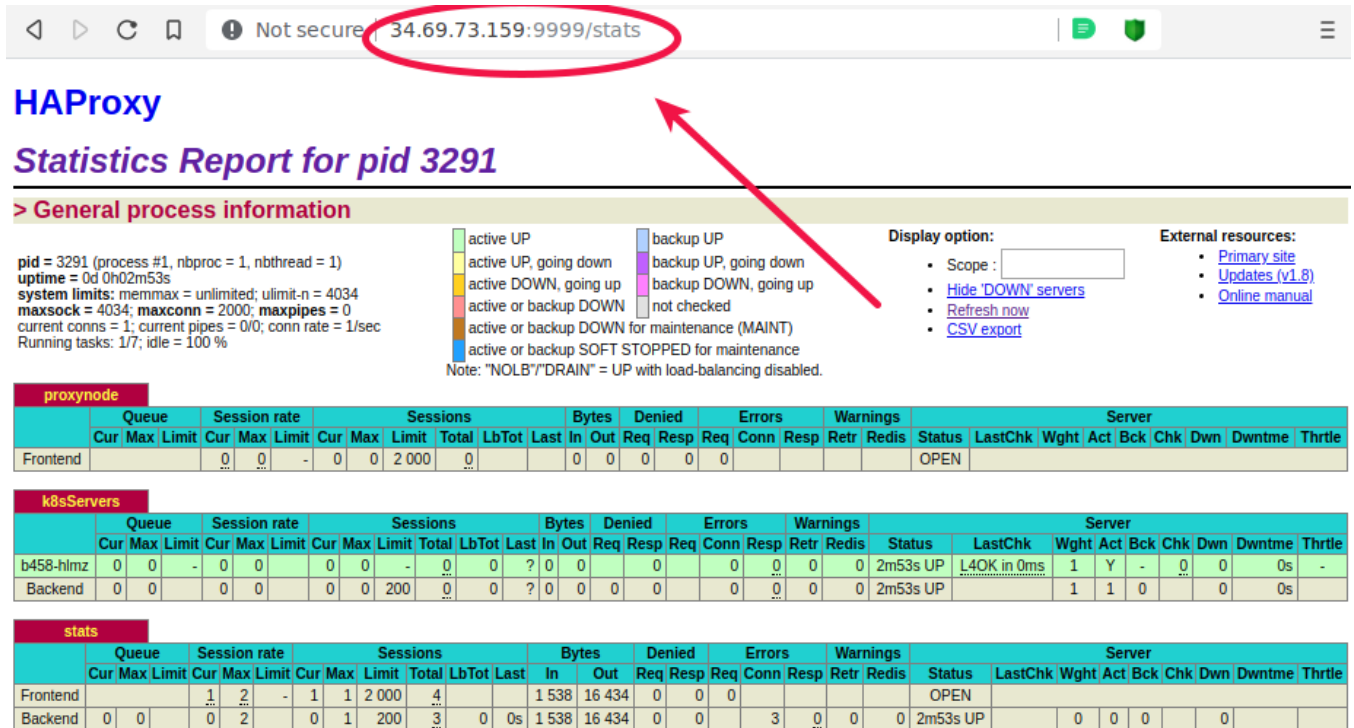
Figure 16.1: **Initial HAProxy Status**

6. Check the node status from the master node then check the proxy statistics. You should see the byte traffic counter increase.

   ```
   student@master:~$ kubectl get nodes
   ```

   ```
   1  NAME               STATUS    ROLES     AGE      VERSION
   2  master   Ready     master    2d6h     v1.19.0
   3  worker            Ready     <none>    2d3h     v1.19.0
   ```

Install Software

We will add two more control planes with stacked **etcd** databases for cluster quorum. You may want to open up two more PuTTY or SSH sessions and color code the terminals to keep track of the nodes.

Initialize the second master before adding the third master

1. Configure and install the kubernetes software on the **second master**. These are the same steps used when we first set up the cluster. The output to each command has been omitted to make the command clear. You may want to copy and paste from the output of **history** to make these steps easier.

```
student@SecondMaster:~$ sudo -i
root@SecondMaster:~$ apt-get update && apt-get upgrade -y
```

2. Install a text editor if not already installed.

```
root@SecondMaster:~$ apt-get install -y vim
```

   (a) **IF** you chose Docker for the master and worker:

```
root@SecondMaster:~$ apt-get install -y docker.io
```

   (b) **IF** you chose cri-o for the master and worker:

```
Please reference the installation lab for detailed installation
and configuration.
```

```
root@SecondMaster:~$ echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" \
        >> /etc/apt/sources.list.d/kubernetes.list
root@SecondMaster:~$ curl -s \
https://packages.cloud.google.com/apt/doc/apt-key.gpg \
| apt-key add -
root@SecondMaster:~$ apt-get update
root@SecondMaster:~$ apt-get install -y \
kubeadm=1.19.0-00 kubelet=1.19.0-00 kubectl=1.19.0-00
root@SecondMaster:~$ apt-mark hold kubelet kubeadm kubectl
root@SecondMaster:~$ exit
```

3. Install the software on the **third master** using the same commands.

Join Master Nodes

1. Edit the `/etc/hosts` file **ON ALL NODES** to ensure the alias of `k8smaster` is set on each node to the proxy IP address. Your IP address may be different.

```
student@master:~$ sudo vim /etc/hosts
```

```
1  10.128.0.64 k8smaster
2  #10.128.0.24 k8smaster
3  127.0.0.1 localhost
4  ....
```

2. On the **first master** create the tokens and hashes necessary to join the cluster. These commands may be in your **history** and easier to copy and paste.

3. Create a new token.

```
student@:~$ sudo kubeadm token create
```

```
1  jasg79.fdh4p279l320cz1g
```

4. Create a new SSL hash.

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

```
1  f62bf97d4fba6876e4c3ff645df3fca969c06169dee3865aab9d0bca8ec9f8cd
```

5. Create a new master certificate to join as a master instead of as a worker.

```
student@:~$ sudo kubeadm init phase upload-certs --upload-certs
```

```
1  [upload-certs] Storing the certificates in Secret "kubeadm-certs" in the "kube-system" Namespace
2  [upload-certs] Using certificate key:
3  5610b6f73593049acddee6b59994360aa4441be0c0d9277c76705d129ba18d65
```

6. On the **second master** use the previous output to build a **kubeadm join** command. Please be aware that multi-line copy and paste from Windows and some MacOS has paste issues. If you get unexpected output copy one line at a time.

```
student@$SecondMaster:~$ sudo kubeadm join k8smaster:6443 \
  --token jasg79.fdh4p279l320cz1g \
  --discovery-token-ca-cert-hash sha256:f62bf97d4fba6876e4c3ff645df3fca969c06169dee3865aab9d0bca8ec9f8cd \
  --control-plane --certificate-key \
  5610b6f73593049acddee6b59994360aa4441be0c0d9277c76705d129ba18d65
```

```
1  [preflight] Running pre-flight checks
2  [WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup driver. The recommended driver \
3     is "systemd". Please follow the guide at https://kubernetes.io/docs/setup/cri/
4  <output_omitted>
```

7. Return to the first master node and check to see if the node has been added and is listed as a `master`.

```
student@master:~$ kubectl get nodes
```

```
1  NAME                STATUS   ROLES    AGE     VERSION
2  master     Ready    master   2d6h    v1.19.0
3  worker        Ready    <none>   2d3h    v1.19.0
4  lfs458-SecondMaster Ready    master   10m     v1.19.0
```

8. Copy and paste the **kubeadm join** command to the third master. Then check that the third master has been added.

```
student@master:~$ kubectl get nodes
```

```
1  NAME                STATUS   ROLES    AGE     VERSION
2  master     Ready    master   2d6h    v1.19.0
3  worker        Ready    <none>   2d3h    v1.19.0
4  lfs458-SecondMaster Ready    master   10m     v1.19.0
5  lfs458-ThirdMaster  Ready    master   3m4s    v1.19.0
```

9. Copy over the configuration file as suggested in the output at the end of the join command. Do this on both newly added master nodes.

```
student@lfs458-SecondMaster$ mkdir -p $HOME/.kube
student@lfs458-SecondMaster$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
student@lfs458-SecondMaster$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

10. On the **Proxy node**. Edit the proxy to include all three master nodes then restart the proxy.

```
student@ha-proxy:~$ sudo vim /etc/haproxy/haproxy.cfg
```

```
1  ....
2  backend k8sServers
3     balance roundrobin
4     server master  10.128.0.24:6443 check
5     server lfs458-SecondMaster  10.128.0.30:6443 check  #<-- Edit/Uncomment these lines
6     server lfs458-ThirdMaster  10.128.0.66:6443 check  #<--
7  ....
```

```
student@ha-proxy:~$ sudo systemctl restart haproxy.service
```

11. View the proxy statistics. When it refreshes you should see three new back-ends. As you check the status of the nodes using **kubectl get nodes** you should see the byte count increase on each node indicating each is handling some of the requests.

proxynode	Queue			Session rate			Sessions						Bytes		Deni
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req
Frontend				0	68	-	11	68	2 000	76			85 805	145 550	0

k8sServers	Queue			Session rate			Sessions						Bytes		Denied	
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp
master1	0	0	-	0	22		5	23	-	26	26	3s	28 029	37 193		0
master2	0	0	-	0	23		4	23	-	25	25	4m6s	26 015	31 374		0
master3	0	0	-	0	23		2	22	-	25	25	10s	31 761	76 983		0
Backend	0	0		0	68		11	68	200	76	76	3s	85 805	145 550	0	0

stats	Queue			Session rate			Sessions						Bytes		Denied	
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp
Frontend				1	2	-	1	1	2 000	7			3 205	56 260	0	
Backend	0	0		0	2		0	1	200	6	0	0s	3 205	56 260	0	

Figure 16.2: **Multiple HAProxy Status**

12. View the logs of the newest **etcd** pod. Leave it running, using the **-f** option in one terminal while running the following commands in a different terminal. As you have copied over the cluster admin file you can run **kubectl** on any master.

student@master:~$ kubectl -n kube-system get pods |grep etcd

```
1  etcd-master                 1/1      Running  0          2d12h
2  etcd-lfs458-SecondMaster            1/1      Running  0          22m
3  etcd-lfs458-ThirdMaster             1/1      Running  0          18m
```

student@master:~$ kubectl -n kube-system logs -f etcd-lfs458-ThirdMaster

```
1  ....
2  2019-08-09 01:58:03.768858 I | mvcc: store.index: compact 300473
3  2019-08-09 01:58:03.770773 I | mvcc: finished scheduled compaction at 300473 (took 1.286565ms)
4  2019-08-09 02:03:03.766253 I | mvcc: store.index: compact 301003
5  2019-08-09 02:03:03.767582 I | mvcc: finished scheduled compaction at 301003 (took 995.775µs)
6  2019-08-09 02:08:03.785807 I | mvcc: store.index: compact 301533
7  2019-08-09 02:08:03.787058 I | mvcc: finished scheduled compaction at 301533 (took 913.185µs)
```

13. Log into one of the **etcd** pods and check the cluster status, using the IP address of each server and port 2379. Your IP addresses may be different. Exit back to the node when done.

student@master:~$ kubectl -n kube-system exec -it etcd-master -- /bin/sh

etcd pod

```
/ # ETCDCTL_API=3 etcdctl -w table \
   --endpoints 10.128.0.66:2379,10.128.0.24:2379,10.128.0.30:2379 \
   --cacert /etc/kubernetes/pki/etcd/ca.crt \
   --cert /etc/kubernetes/pki/etcd/server.crt \
```

```
      --key /etc/kubernetes/pki/etcd/server.key \
      endpoint status
 1  +-----------------+------------------+---------+---------+-----------+-----------+-------------+
 2  |    ENDPOINT     |        ID        | VERSION | DB SIZE | IS LEADER | RAFT TERM | RAFT INDEX |
 3  +-----------------+------------------+---------+---------+-----------+-----------+-------------+
 4  | 10.128.0.66:2379 | 2331065cd4fb02ff |  3.3.10 |  24 MB  |      true |        11 |      392573 |
 5  | 10.128.0.24:2379 | d2620a7d27a9b449 |  3.3.10 |  24 MB  |     false |        11 |      392573 |
 6  | 10.128.0.30:2379 | ef44cc541c5f37c7 |  3.3.10 |  24 MB  |     false |        11 |      392573 |
 7  +-----------------+------------------+---------+---------+-----------+-----------+-------------+
```

Test Failover

Now that the cluster is running and has chosen a leader we will shut down docker, which will stop all containers on that node. This will emulate an entire node failure. We will then view the change in leadership and logs of the events.

1. If you used Docker, Shut down the service on the node which shows `IS LEADER` set to `true`.

   ```
   student@master:~$ sudo systemctl stop docker.service
   ```

   If you chose cri-o as the container engine then the cri-o service and conmon processes are distinct. It may be easier to reboot the node and refresh the HAProxy web page until it shows the node is down. It may take a while for the node to finish the boot process. The second and third master should work the entire time.

   ```
   student@master:~$ sudo reboot
   ```

2. You will probably note the **logs** command exited when the service shut down. Run the same command and, among other output, you'll find errors similar to the following. Note the messages about losing the leader and electing a new one, with an eventual message that a peer has become inactive.

   ```
   student@master:~$ kubectl -n kube-system logs -f etcd-lfs458-ThirdMaster
   ```

```
 1  ....
 2  2019-08-09 02:11:39.569827 I | raft: 2331065cd4fb02ff [term: 9] received a MsgVote message with higher \
 3                                  term from ef44cc541c5f37c7 [term: 10]
 4  2019-08-09 02:11:39.570130 I | raft: 2331065cd4fb02ff became follower at term 10
 5  2019-08-09 02:11:39.570148 I | raft: 2331065cd4fb02ff [logterm: 9, index: 355240, vote: 0] cast MsgVote \
 6                                  for ef44cc541c5f37c7 [logterm: 9, index: 355240] at term 10
 7  2019-08-09 02:11:39.570155 I | raft: raft.node: 2331065cd4fb02ff lost leader d2620a7d27a9b449 at term 10
 8  2019-08-09 02:11:39.572242 I | raft: raft.node: 2331065cd4fb02ff elected leader ef44cc541c5f37c7 at \
 9                                  term 10
10  2019-08-09 02:11:39.682319 W | rafthttp: lost the TCP streaming connection with peer d2620a7d27a9b449 \
11                                  (stream Message reader)
12  2019-08-09 02:11:39.682635 W | rafthttp: lost the TCP streaming connection with peer d2620a7d27a9b449 \
13                                  (stream MsgApp v2 reader)
14  2019-08-09 02:11:39.706068 E | rafthttp: failed to dial d2620a7d27a9b449 on stream MsgApp v2 \
15                                  (peer d2620a7d27a9b449 failed to find local node 2331065cd4fb02ff)
16  2019-08-09 02:11:39.706328 I | rafthttp: peer d2620a7d27a9b449 became inactive (message send to peer failed)
17  ....
```

3. View the proxy statistics. The proxy should show the first master as down, but the other master nodes remain up.

k8sServers

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Status	LastChk	W
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	W
master1	0	0	-	0	22		0	23	-	173	129	12m18s	11 110 233	62 695 354		0		0	19	44	0	12m DOWN	L4CON in 0ms	
master2	0	0	-	0	23		6	23	-	129	129	12m6s	299 280	2 703 547		0		0	0	0	0	4h15m UP	L4OK in 0ms	
master3	0	0	-	0	23		5	22	-	128	128	12m23s	362 790	6 078 463		0		0	1	0	0	4h15m UP	L4OK in 0ms	
Backend	0	0		0	68		11	68	200	387	386	12m6s	11 772 303	71 477 364	0	0		0	20	44	0	4h15m UP		

stats

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Status	LastChk	Wght
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght
Frontend				1	2	-	1	1	2 000	10			4 885	93 693	0	0	0					OPEN		
Backend	0	0		0	2		0	1	200	9	0	0s	4 885	93 693	0	0		9	0	0	0	4h15m UP		0

Figure 16.3: **HAProxy Down Status**

4. View the status using **etcdctl** from within one of the running **etcd** pods. You should get an error for the endpoint you shut down and a new leader of the cluster.

```
student@lfs458-SecondMaster:~$ kubectl -n kube-system exec -it etcd-lfs458-SecondMaster -- /bin/sh
```

etcd pod

```
/ # ETCDCTL_API=3 etcdctl -w table \
    --endpoints 10.128.0.66:2379,10.128.0.24:2379,10.128.0.30:2379 \
    --cacert /etc/kubernetes/pki/etcd/ca.crt \
    --cert /etc/kubernetes/pki/etcd/server.crt \
    --key /etc/kubernetes/pki/etcd/server.key \
    endpoint status
```

```
1  Failed to get the status of endpoint 10.128.0.66:2379 (context deadline exceeded)
2  +------------------+------------------+---------+---------+-----------+-----------+------------+
3  |     ENDPOINT     |        ID        | VERSION | DB SIZE | IS LEADER | RAFT TERM | RAFT INDEX |
4  +------------------+------------------+---------+---------+-----------+-----------+------------+
5  | 10.128.0.24:2379 | d2620a7d27a9b449 | 3.3.10  |  24 MB  |      true |        12 |     395729 |
6  | 10.128.0.30:2379 | ef44cc541c5f37c7 | 3.3.10  |  24 MB  |     false |        12 |     395729 |
7  +------------------+------------------+---------+---------+-----------+-----------+------------+
```

5. Turn the docker service back on. You should see the peer become active and establish a connection.

```
student@master:~$ sudo systemctl start docker.service
```

```
student@master:~$ kubectl -n kube-system logs -f etcd-lfs458-ThirdMaster
```

```
1  ....
2  2019-08-09 02:45:11.337669 I | rafthttp: peer d2620a7d27a9b449 became active
3  2019-08-09 02:45:11.337710 I | rafthttp: established a TCP streaming connection with peer\
4                                 d2620a7d27a9b449 (stream MsgApp v2 reader)
5  ....
```

6. View the **etcd** cluster status again. Experiment with how long it takes for the **etcd** cluster to notice failure and choose a new leader with the time you have left.

Appendices

Appendix A

Domain Review

A.1   Exam Domain Review

✎ Exercise A.1: Are you Ready?

1. If you do not have tested, working YAML examples for all of the objects the domain review mentions bookmarked in the browser you are allowed to use for the exam you may:

   a. Run out of time while searching for the yaml.

   b. Make more mistakes.

   c. Use YAML that isn't proper for the Kubernetes version of the exam and waste time trying to troubleshoot

   d. All of the above.

2. Answer all that apply. In the context of the `Curriculum Overview` the term **Understand** when stating what a candidate should be able to do means:

   a. You only have a general idea what the object does.

   b. You can create the object.

   c. You can configure and integrate the object with other objects.

   d. You can properly update and test the object.

   e. You can troubleshoot the object.

3. Have you practiced creating, integrating, and troubleshooting all of the domain review items **at speed**?

   a. No. I kind of did the exercises. So I'm good.

   b. No. I did the labs twice over a two week period.

   c. Yes. I know the exam is intense and practiced with a clock running to make sure I can get everything done and also check my work.

✅ Solution A.1

Are You Ready?

1. d.

2. b, c, d, e.

3. Hopefully c.

✎ Exercise A.2: Preparing for the CKA Exam

⚠ **Very Important**

The source pages and content in this review could change at any time. **IT IS YOUR RESPONSIBILITY TO CHECK THE CURRENT INFORMATION**.

Before Taking Exam

Use this exercise as a resource after you complete the course but before you take the exam. Review the resources, build your YAML bookmarks, and practice creating and working with objects at exam speed to assist with review and preparation.

1. Using a browser go to https://www.cncf.io/certification/cka/ and read through the program description.

2. In the **Exam Resources** section open the Curriculum Overview and Candidate-handbook in new tabs. Both of these should be read and understood prior to sitting for the exam.

3. Navigate to the Curriculum Overview tab. You should see links for domain information for various versions of the exam. Select the latest version, such as **CKA_Curriculum_V1.19.pdf**. The versions you see may be different. You should see a new page showing a PDF.

4. Read through the document. Be aware that the term `Understand`, such as `Understand Services`, is more than just knowing they exist. In this case expect it to also mean create, configure, update, and troubleshoot.

5. Using only the exam-allowed browser, URLs, and sub-domains search for and bookmark YAML examples for each item. Ensure it works for the version of the exam you are taking, as the YAML may not have been re-tested after a new release. URLs may change, plan on checking each book mark prior to taking the exam.

6. Using a timer and bookmarked YAML files see how long it takes you to create and verify the objects listed below. Write down the time. Try it again and see how much faster you can complete and test each step

> "Practice until you get it right. Then practice until you can't get it wrong" -Unknown

Domain Review Items

This list is copied from competency domains found on the PDF, and can be used as a checklist to ensure you have all the necessary YAML files and resources bookmarked. Again, it remains your responsibility to check the web page for any changes to this list.

- **Cluster Architecture, Installation & Configuration**
    - Manage role based access control (RBAC)
    - Use Kubeadm to install a basic cluster
    - Manage a highly-available Kubernetes cluster
    - Provision underlying infrastructure to deploy a Kubernetes cluster

- – Preform a version upgrade on a Kubernetes cluster using Kubeadm
- – Implement etcd backup and restore

- **Workloads & Scheduling**

  - – Understand deployments and how to preform rolling updates and rollbacks
  - – Use ConfigMaps and Secrets to configure applications
  - – Know how to scale applications
  - – Understand the primitives used to create robust, self-healing, application deployments
  - – Understand how resource limits can affect Pod scheduling
  - – Awareness of manifest management and common templating tools

- **Services & Networking**

  - – Understand host networking configuration on the cluster nodes
  - – Understand connectivity between Pods
  - – Understand ClusterIP, NodePort, LoadBalancer service types and endpoints
  - – Know how to use Ingress controllers and Ingress resources
  - – Know how to configure and use CoreDNS
  - – Choose an appropriate container network interface plugin

- **Storage**

  - – Understand storage classes, persistent volumes
  - – Understand volume mode, access modes and reclaim policies for volumes
  - – Understand persistent volume claims primitive
  - – Know how to configure applications with persistent storage

- **Troubleshooting**

  - – Evaluate cluster and node logging
  - – Understand how to monitor applications
  - – Manage container stdout & stderr logs
  - – Troubleshoot application failure
  - – Troubleshoot cluster component failure
  - – Troubleshoot networking

✎ Exercise A.3: Practicing Skills

This exercise is to help you practice your skills. It does not cover all the items listed in the domain review guide. You should develop your own steps to build a full list of skill tests and steps.

Also note that all the detailed steps are not included. You should be able to complete these steps without being told what to type.

Remember to bookmark any YAML example files you need, using the three URL locations allowed by the exam.

1. Find and use the `review1.yaml` file included in the course tarball. Use the **find** output and copy the YAML file to your home directory. Use **kubectl create** to create the object. Determine if the pod is running. Fix any errors you may encounter. The use of **kubectl describe** may be helpful.

   ```
   student@ckad-1:~$ find ~ -name review1.yaml

   student@ckad-1:~$ cp <copy-paste-from-above>  .

   student@ckad-1:~$ kubectl create -f review1.yaml
   ```

2. After you get the pod running remove any pods or services you may have created as part of the review before moving on to the next section. For example:

   ```
   student@ckad-1:~$ kubectl delete -f review1.yaml
   ```

3. Use the `review2.yaml` file to create a non-working deployment. Fix the deployment such that both containers are running and in a `READY` state. The web server listens on port 80, and the proxy listens on port 8080.

4. View the default page of the web server. When successful verify the `GET` activity logs in the container log. The message should look something like the following. Your time and IP may be different.

   ```
   192.168.124.0 - - [3/Dec/2020:03:30:31 +0000] "GET / HTTP/1.1" 200 612 "-"
   "curl/7.58.0" "-"
   ```

5. Find and use the `review4.yaml` file to create a pod, and verify it's running

6. Edit the pod such that it only runs on your worker node using the `nodeSelector` label.

7. Determine the CPU and memory resource requirements of `design-pod1`.

8. Edit the pod resource requirements such that the CPU limit is exactly twice the amount requested by the container. (Hint: subtract .22)

9. Increase the memory resource limit of the pod until the pod shows a `Running` status. This may require multiple edits and attempts. Determine the minimum amount necessary for the `Running` status to persist at least a minute.

10. Use the `review5.yaml` file to create several pods with various labels.

11. Using **only** the –selector value `tux` to delete only those pods. This should be half of the pods. Hint, you will need to view pod settings to determine the key value as well.

12. Create a new cronjob which runs `busybox` and the `sleep 30` command. Have the cronjob run every three minutes. View the job status to check your work. Change the settings so the pod runs 10 minutes from the current time, every week. For example, if the current time was 2:14PM, I would configure the job to run at 2:24PM, every Monday.

13. Delete any objects created during this review. You may want to delete all but the `cronjob` if you'd like to see if it runs in 10 minutes. Then delete that object as well.

14. Create a new secret called `specialofday` using the key `entree` and the value `meatloaf`.

15. Create a new deployment called `foodie` running the `nginx` image.

16. Add the `specialofday` secret to pod mounted as a volume under the `/food/` directory.

17. Execute a bash shell inside a `foodie` pod and verify the secret has been properly mounted.

18. Update the deployment to use the `nginx:1.12.1-alpine` image and verify the new image is in use.

19. Roll back the deployment and verify the typical, current stable version of nginx is in use again.

20. Create a new 200M NFS volume called `reviewvol` using the NFS server configured earlier in the lab.

21. Create a new PVC called `reviewpvc` which will uses the `reviewvol` volume.

22. Edit the deployment to use the PVC and mount the volume under `/newvol`

23. Execute a bash shell into the nginx container and verify the volume has been mounted.

24. Delete any resources created during this review.

25. Create a new deployment which uses the `nginx` image.

26. Create a new `LoadBalancer` service to expose the newly created deployment. Test that it works.

27. Create a new NetworkPolicy called `netblock` which blocks all traffic to pods in this deployment only. Test that all traffic is blocked to deployment.

28. Create a pod running nginx and ensure traffic can reach that deployment.

29. Update the `netblock` policy to allow traffic to the pod on port 80 only. Test that you can now access the default nginx web page.

30. Find and use the `review6.yaml` file to create a pod.

    `student@ckad-1:~$ kubectl create -f review6.yaml`

31. View the status of the pod.

32. Use the following commands to figure out why the pod has issues.

    `student@ckad-1:~$ kubectl get pod securityreview`

    `student@ckad-1:~$ kubectl describe pod securityreview`

    `student@ckad-1:~$ kubectl logs securityreview`

33. After finding the errors, log into the container and find the proper id of the nginx user.

34. Edit the pod such that the `securityContext` is in place and allows the web server to read the proper configuration files.

35. Create a new `serviceAccount` called `securityaccount`.

36. Create a ClusterRole named `secrole` which only allows create, delete, and list of pods in all apiGroups.

37. Bind the new clusterRole to the new serviceAccount.

38. Locate the token of the `securityaccount`. Create a file called `/tmp/securitytoken`. Put only the value of `token:` is equal to, a long string that may start with eyJh and be several lines long. Careful that only that string exists in the file.

39. Remove any resources you have added during this review

40. Create a new pod called `webone`, running the `nginx` service. Expose port 80.

41. Create a new service named `webone-svc`. The service should be accessible from outside the cluster.

42. Update both the pod and the service with selectors so that traffic for to the service IP shows the web server content.

43. Change the type of the service such that it is only accessible from within the cluster. Test that exterior access no longer works, but access from within the node works.

44. Deploy another pod, called `webtwo`, this time running the `wlniao/website` image. Create another service, called `webtwo-svc` such that only requests from within the cluster work. Note the default page for each server is distinct.

45. Test DNS names and verify CoreDNS is properly functioning.

46. Install and configure an ingress controller such that requests for webone.com see the `nginx` default page, and requests for webtwo.org see the `wlniao/website` default page. It does not matter which ingress controller you use.

47. Remove any resources created in this review.

48. Install a new cluster using an recent, previous version of Kubernetes. Backup etcd, then properly upgrade the entire cluster.

49. Create a pod running `busybox` without the scheduler being consulted.

50. Continue to create objects, integrate them with other objects and troubleshoot until each domain item has been covered.