



Kubernetes Fundamentals

Version 2019-12-03



Version 2019-12-03

© Copyright the Linux Foundation 2019. All rights reserved.

© Copyright the Linux Foundation 2019. All rights reserved.

The training materials provided or developed by The Linux Foundation in connection with the training services are protected by copyright and other intellectual property rights.

Open source code incorporated herein may have other copyright holders and is used pursuant to the applicable open source license.

The training materials are provided for individual use by participants in the form in which they are provided. They may not be copied, modified, distributed to non-participants or used to provide training to others without the prior written consent of The Linux Foundation.

No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without express prior written consent.

Published by:

the **Linux Foundation**

<https://www.linuxfoundation.org>

No representations or warranties are made with respect to the contents or use of this material, and any express or implied warranties of merchantability or fitness for any particular purpose or specifically disclaimed.

Although third-party application software packages may be referenced herein, this is for demonstration purposes only and shall not constitute an endorsement of any of these software applications.

Linux is a registered trademark of Linus Torvalds. Other trademarks within this course material are the property of their respective owners.

If there are any questions about proper and fair use of the material herein, please contact:

training@linuxfoundation.org

Contents

- 1 Introduction 1**
 - 1.1 Labs 1
- 2 Basics of Kubernetes 3**
 - 2.1 Labs 3
- 3 Installation and Configuration 5**
 - 3.1 Labs 5
- 4 Kubernetes Architecture 23**
 - 4.1 Labs 23
- 5 APIs and Access 31**
 - 5.1 Labs 31
- 6 API Objects 37**
 - 6.1 Labs 37
- 7 Managing State With Deployments 47**
 - 7.1 Labs 47
- 8 Services 55**
 - 8.1 Labs 55
- 9 Volumes and Data 61**
 - 9.1 Labs 61
- 10 Ingress 77**
 - 10.1 Labs 77
- 11 Scheduling 83**
 - 11.1 Labs 83
- 12 Logging and Troubleshooting 91**
 - 12.1 Labs 91
- 13 Custom Resource Definition 99**
 - 13.1 Labs 99
- 14 Helm 103**

14.1 Labs	103
15 Security	109
15.1 Labs	109
16 High Availability	117
16.1 Labs	117

List of Figures

3.1	External Access via Browser	21
10.1	Accessing the API	82
12.1	External Access via Browser	97
12.2	External Access via Browser	98
12.3	External Access via Browser	98
16.1	Initial HAProxy Status	119
16.2	Multiple HAProxy Status	122
16.3	HAProxy Down Status	123

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:

```
$ 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 `is:issue is:open:` which allows you to filter on the kind of information you would like to see. Append the search string to read: `is:issue 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.
7. Explore the page with the remaining time left.

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 **Google Cloud Platform (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 v1.16.1 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.

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.



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_V2019-12-03_SOLUTIONS.tar.bz2 \
    --user=LFtraining --password=Penguin2014
```

```
$ tar -xvf LFS258_V2019-12-03_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
```

```
The authenticity of host '54.214.214.156 (35.226.100.87)' can't be established.
ECDSA key fingerprint is SHA256:IPvznbkx93/Wc+ACwXrCcDDgvBwmvEXC9vmYhk2Wo1E.
ECDSA key fingerprint is MD5:d8:c9:4b:b0:b0:82:d3:95:08:08:4a:74:1b:f6:e1:9f.
Are you sure you want to continue connecting (yes/no)? yes
```

```
Warning: Permanently added '35.226.100.87' (ECDSA) to the list of known hosts.
<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@lfs458-node-1a0a:~$ sudo -i
```

```
root@lfs458-node-1a0a:~# apt-get update && apt-get upgrade -y
```

```
<output_omitted>
```

You can choose this option to avoid being prompted; instead, all necessary restarts will be done for you automatically so you can avoid being asked questions on each library upgrade.

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

```
<output_omitted>
```

A new version (/tmp/fileEbke6q) of configuration file /etc/ssh/sshd_config is available, but the version installed currently has been locally modified.

1. install the package maintainer's version
2. keep the local version currently installed
3. show the differences between the versions
4. show a side-by-side difference between the versions
5. show a 3-way difference between available versions
6. do a 3-way merge between available versions
7. start a new shell to examine the situation

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

```
<output_omitted>
```

3. The main choices for a container environment are **Docker** and **cri-o**. We will use **Docker** for class, as **cri-o** is not yet the default when building the cluster with **kubeadm**. You should also make sure you have a text editor like **nano**, **vim**, or **emacs**. Any will do, the labs use a popular option, **vim**.

```
root@lfs458-node-1a0a:~# apt-get install -y docker.io
```

```
<output-omitted>
```

```
root@lfs458-node-1a0a:~# apt-get install -y vim
```

```
<output-omitted>
```

4. 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@lfs458-node-1a0a:~# vim /etc/apt/sources.list.d/kubernetes.list
```

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

5. 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@lfs458-node-1a0a:~# curl -s \
    https://packages.cloud.google.com/apt/doc/apt-key.gpg \
    | apt-key add -
```

```
OK
```

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

```
root@lfs458-node-1a0a:~# apt-get update
<output-omitted>
```

- 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 version we install.

```
root@lfs458-node-1a0a:~# apt-get install -y \
    kubeadm=1.16.1-00 kubelet=1.16.1-00 kubect1=1.16.1-00
<output-omitted>

root@lfs458-node-1a0a:~# apt-mark hold kubelet kubeadm kubect1
kubelet set on hold.
kubeadm set on hold.
kubect1 set on hold.
```

- 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. The 3.3 version of **Calico** has more than one configuration file for flexibility with RBAC. Download the configuration files for Calico and RBAC. Once downloaded look for the expected IPV4 range for containers to use in the configuration file.

A short url for each file is shown in the following **wget** commands. the longer URLs can be found here: <https://docs.projectcalico.org/v3.3/getting-started/kubernetes/installation/hosted/rbac-kdd.yaml> and: <https://docs.projectcalico.org/v3.9/manifests/calico.yaml>

```
root@lfs458-node-1a0a:~# wget https://tinyurl.com/yb4xturm -O rbac-kdd.yaml
root@lfs459-node-1a0a:~# wget https://tinyurl.com/y2vqsobb -O calico.yaml
```

- 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.

```
root@lfs458-node-1a0a:~# less calico.yaml
```

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  ....
```

- 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@lfs458-node-1a0a:~# ip addr show
```

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

11. 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@lfs458-node-1a0a:~# vim /etc/hosts

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

12. 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@lfs458-node-1a0a:~# vim kubeadm-config.yaml
```



kubeadm-config.yaml

```
1 apiVersion: kubeadm.k8s.io/v1beta2
2 kind: ClusterConfiguration
3 kubernetesVersion: 1.16.1           #<-- 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
```

13. 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@lfs458-node-1a0a:~# kubeadm init --config=kubeadm-config.yaml --upload-certs \
| tee kubeadm-init.out      # Save output for future review
```



Please Note

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

```
[init] Using Kubernetes version: v1.16.1
[preflight] Running pre-flight checks
[WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the
Docker cgroup driver. The recommended driver is "systemd".
....
```

You can now join any number of the control-plane node running the following command on each as root:

```
kubeadm join k8smaster:6443 --token vapzqi.et2p9zbkzk29wwth \
```

```
--discovery-token-ca-cert-hash sha256:f62bf97d4fba6876e4c3ff645df3fca969c06169dee3865aab9d0bca8ec9f8cd \
--control-plane --certificate-key 911d41fcada89a18210489afaa036cd8e192b1f122ebb1b79cce1818f642fab8
```

Please note that the certificate-key gives access to cluster sensitive data, keep it secret!
As a safeguard, uploaded-certs will be deleted in two hours; If necessary, you can use
"kubeadm init phase upload-certs --upload-certs" to reload certs afterward.

Then you can join any number of worker nodes by running the following on each as root:

```
kubeadm join k8smaster:6443 --token vapzqi.et2p9zbkzk29wwth \
--discovery-token-ca-cert-hash sha256:f62bf97d4fba6876e4c3ff645df3fca969c06169dee3865aab9d0bca8ec9f8cd
```

14. 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@lfs458-node-1a0a:~# exit

logout

student@lfs458-node-1a0a:~$ mkdir -p $HOME/.kube

student@lfs458-node-1a0a:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

student@lfs458-node-1a0a:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config

student@lfs458-node-1a0a:~$ less .kube/config

apiVersion: v1
clusters:
- cluster:
<output_omitted>
```

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

```
student@lfs458-node-1a0a:~$ sudo cp /root/rbac-kdd.yaml .

student@lfs458-node-1a0a:~$ kubectl apply -f rbac-kdd.yaml

clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created

student@lfs458-node-1a0a:~$ sudo cp /root/calico.yaml .

student@lfs458-node-1a0a:~$ kubectl apply -f calico.yaml

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

16. 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 `/.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@lfs458-node-1a0a:~$ sudo apt-get install bash-completion -y

<exit and log back in>

student@lfs458-node-1a0a:~$ source <(kubectl completion bash)

student@lfs458-node-1a0a:~$ echo "source <(kubectl completion bash)" >> ~/.bashrc
```

17. 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@lfs458-node-1a0a:~$ kubectl des<Tab> n<Tab><Tab> lfs458-<Tab>

student@lfs458-node-1a0a:~$ kubectl -n kube-s<Tab> g<Tab> po<Tab>
```

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

```
student@lfs458-node-1a0a:~$ sudo kubeadm config print init-defaults
apiVersion: kubeadm.k8s.io/v1beta2
bootstrapTokens:
- groups:
  - system:bootstrappers:kubeadm:default-node-token
  token: abcdef.0123456789abcdef
  ttl: 24h0m0s
  usages:
  - signing
  - authentication
kind: InitConfiguration
<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 **lfs458-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.

```
student@lfs458-worker:~$ sudo -i

root@lfs458-worker:~# apt-get update && apt-get upgrade -y

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

root@lfs458-worker:~# apt-get install -y docker.io vim

root@lfs458-worker:~# vim /etc/apt/sources.list.d/kubernetes.list

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

```

root@lfs458-worker:~# curl -s \
    https://packages.cloud.google.com/apt/doc/apt-key.gpg \
    | apt-key add -

root@lfs458-worker:~# apt-get update

root@lfs458-worker:~# apt-get install -y \
    kubeadm=1.16.1-00 kubelet=1.16.1-00 kubectl=1.16.1-00

root@lfs458-worker:~# apt-mark hold kubeadm kubelet kubectl

```

- 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@lfs458-node-1a0a:~$ ip addr show ens4 | grep inet

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

```

- 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@lfs458-node-1a0a:~$ sudo kubeadm token list

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

```

- 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@lfs458-node-1a0a:~$ sudo kubeadm token create

27eee4.6e66ff60318da929

```

- 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@lfs458-node-1a0a:~$ openssl x509 -pubkey \
    -in /etc/kubernetes/pki/ca.crt | openssl rsa \
    -pubin -outform der 2>/dev/null | openssl dgst \
    -sha256 -hex | sed 's/^.* //'

6d541678b05652e1fa5d43908e75e67376e994c3483d6683f2a18673e5d2a1b0

```

- 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@lfs458-worker:~# vim /etc/hosts

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

```

- 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@lfs458-worker:~# kubeadm join \
    --token 27eee4.6e66ff60318da929 \
    k8smaster:6443 \
    --discovery-token-ca-cert-hash \
    sha256:6d541678b05652e1fa5d43908e75e67376e994c3483d6683f2a18673e5d2a1b0

```

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

This node has joined the cluster:

- * Certificate signing request was sent to apiservert and a response was received.
- * The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

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@lfs458-worker:~# exit

student@lfs458-worker:~$ kubectl get nodes
The connection to the server localhost:8080 was refused
- did you specify the right host or port?

student@lfs458-worker:~$ ls -l .kube
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@lfs458-node-1a0a:~$ kubectl get node

NAME                STATUS    ROLES    AGE    VERSION
lfs458-worker       Ready    <none>   50s    v1.16.1
lfs458-node-1a0a    Ready    master   28m    v1.16.1
```

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@lfs458-node-1a0a:~$ kubectl describe node lfs458-node-1a0a
Name:                lfs458-node-1a0a
Roles:               master
Labels:              beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/hostname=lfs458-node-1a0a
                    node-role.kubernetes.io/master=
Annotations:         kubeadm.alpha.kubernetes.io/cri-socket: /var/run/dockershim.sock
                    node.alpha.kubernetes.io/ttl: 0
                    projectcalico.org/IPV4Address: 10.142.0.3/32
                    volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp:   Mon, 07 Jan 2019 22:04:03 +0000
Taints:              node-role.kubernetes.io/master:NoSchedule
<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@lfs458-node-1a0a:~$ kubectl describe node | grep -i taint
```

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

```
student@lfs458-node-1a0a:~$ kubectl taint nodes \
--all node-role.kubernetes.io/master-
```

```
node/lfs458-node-1a0a untainted
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@lfs458-node-1a0a:~$ kubectl describe node | grep -i taint
```

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

```
student@lfs458-node-1a0a:~$ kubectl taint nodes \
--all node.kubernetes.io/not-ready-
```

```
node/lfs58-node-1a0a untainted
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@lfs458-node-1a0a:~$ kubectl get pods --all-namespaces
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
kube-system	calico-etcd-jlgwr	1/1	Running	0	6m
kube-system	calico-kube-controllers-74b888b647-wlqf5	1/1	Running	0	6m
kube-system	calico-node-tpvnr	2/2	Running	0	6m
kube-system	coredns-78fcdf6894-nc5cn	1/1	Running	0	17m
kube-system	coredns-78fcdf6894-xs96m	1/1	Running	0	17m

<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@lfs458-node-1a0a:~$ kubectl get pods --all-namespaces
```

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

<output_omitted>

```
student@lfs458-node-1a0a:~$ kubectl -n kube-system delete \
pod coredns-576cbf47c7-vq5dz coredns-576cbf47c7-rn6v4
```

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

- 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@lfs458-node-1a0a:~$ ip a
<output_omitted>
4: tunl0@NONE: <NOARP,UP,LOWER_UP> mtu 1440 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
    inet 192.168.0.1/32 brd 192.168.0.1 scope global tunl0
        valid_lft forever preferred_lft forever
6: calib0b93ed4661@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu
1440 qdisc noqueue state UP group default
    link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netnsid 1
    inet6 fe80::ecee:eeff:feee:eeee/64 scope link
        valid_lft forever preferred_lft forever
<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.

- 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@lfs458-node-1a0a:~$ kubectl create deployment nginx --image=nginx
deployment.apps/nginx created
```

```
student@lfs458-node-1a0a:~$ kubectl get deployments
```

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

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

```
student@lfs458-node-1a0a:~$ kubectl describe deployment nginx
Name:                nginx
Namespace:           default
CreationTimestamp:    Mon, 23 Apr 2019 22:38:32 +0000
Labels:              app=nginx
Annotations:         deployment.kubernetes.io/revision: 1
Selector:             app=nginx
Replicas:            1 desired | 1 updated | 1 total | 1 ava...
StrategyType:        RollingUpdate
MinReadySeconds:     0
RollingUpdateStrategy: 25% max unavailable, 25% max surge
<output_omitted>
```

- 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@lfs458-node-1a0a:~$ kubectl get events
<output_omitted>
```

- 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@lfs458-node-1a0a:~$ 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 40, if others have already been removed.

```
student@lfs458-node-1a0a:~$ kubectl get deployment nginx -o yaml > first.yaml
```

```
student@lfs458-node-1a0a:~$ vim first.yaml
```

<Remove the lines mentioned above>

6. Delete the existing deployment.

```
student@lfs458-node-1a0a:~$ kubectl delete deployment nginx
deployment.apps "nginx" deleted
```

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

```
student@lfs458-node-1a0a:~$ kubectl create -f first.yaml
deployment.apps/nginx created
```

8. Look at the yaml output of this iteration and compare it against the first. The time stamp, resource version and uid 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. The status should not be hard-coded either.

```
student@lfs458-node-1a0a:~$ kubectl get deployment nginx -o yaml > second.yaml
```

```
student@lfs458-node-1a0a:~$ diff first.yaml second.yaml
```

<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@lfs458-node-1a0a:~$ kubectl create deployment two --image=nginx --dry-run -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@lfs458-node-1a0a:~$ kubectl get deployment
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
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. Note there is more detail to the **--export** option. The flag has been **deprecated** and may be removed in the future.

```
student@lfs458-node-1a0a:~$ kubectl get deployments nginx --export -o yaml
```

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@lfs458-node-1a0a:~$ kubectl get deployment nginx --export -o json
```

JSON

```
1 {
2   "apiVersion": "extensions/v1beta1",
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@lfs458-node-1a0a:~$ kubectl expose -h
```

```
<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@lfs458-node-1a0a:~$ kubectl expose deployment/nginx
```

```
error: couldn't find port via --port flag or introspection
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@lfs458-node-1a0a:~$ vim first.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@lfs458-node-1a0a:~$ kubectl replace -f first.yaml
deployment.apps/nginx replaced

```

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

```

student@lfs458-node-1a0a:~$ kubectl get deploy,pod

```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/nginx	1/1	1	1	2m4s

NAME	READY	STATUS	RESTARTS	AGE
pod/nginx-7db75b8b78-qjffm	1/1	Running	0	8s

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

```

student@lfs458-node-1a0a:~$ kubectl expose deployment/nginx
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@lfs458-node-1a0a:~$ kubectl get svc nginx

```

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


```

student@lfs458-node-1a0a:~$ kubectl get ep nginx

```

NAME	ENDPOINTS	AGE
nginx	192.168.1.5:80	26s

19. Determine which node the container is running on. Log into that node and use `tcpdump` to view traffic on the `tunl0`, 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@lfs458-node-1a0a:~$ kubectl describe pod nginx-7cbc4b4d9c-d27xw \
| grep Node:
Node:    lfs458-worker/10.128.0.5

student@lfs458-worker:~$ sudo tcpdump -i tunl0

tcpdump: verbose output suppressed, use -v or -vv for full protocol...
listening on tunl0, link-type EN10MB (Ethernet), capture size...
<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@lfs458-node-1a0a:~$ curl 10.100.61.122:80
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
<output_omitted>
```

```
student@lfs458-node-1a0a:~$ curl 192.168.1.5:80
```

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

```
student@lfs458-node-1a0a:~$ kubectl get deployment nginx
```

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

```
student@lfs458-node-1a0a:~$ kubectl scale deployment nginx --replicas=3
```

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

```
student@lfs458-node-1a0a:~$ kubectl get deployment nginx
```

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

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@lfs458-node-1a0a:~$ kubectl get ep nginx
```

NAME	ENDPOINTS	AGE
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@lfs458-node-1a0a:~$ kubectl get po -o wide
```

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

```
student@lfs458-node-1a0a:~$ kubectl delete po nginx-1423793266-7f1qw
```

```
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.

```
student@lfs458-node-1a0a:~$ kubectl get po
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-1423793266-13p69	1/1	Running	0	9s
nginx-1423793266-8w2nk	1/1	Running	0	4m1s
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@lfs458-node-1a0a:~$ kubectl get ep nginx
```

NAME	ENDPOINTS	AGE
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@lfs458-node-1a0a:~$ curl 10.100.61.122:80
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body
<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@lfs458-node-1a0a:~$ kubectl get po
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-1423793266-13p69	1/1	Running	0	4m10s
nginx-1423793266-8w2nk	1/1	Running	0	8m2s
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@lfs458-node-1a0a:~$ kubectl exec nginx-1423793266-13p69 \
-- printenv |grep KUBERNETES
```

```
KUBERNETES_SERVICE_PORT=443
KUBERNETES_SERVICE_HOST=10.96.0.1
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT=tcp://10.96.0.1:443
<output_omitted>
```

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

```
student@lfs458-node-1a0a:~$ kubectl get svc
```

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

4. Delete the service.

```
student@lfs458-node-1a0a:~$ kubectl delete svc nginx
```

```
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@lfs458-node-1a0a:~$ kubectl expose deployment nginx --type=LoadBalancer
service/nginx exposed
```

```
student@lfs458-node-1a0a:~$ kubectl get svc
```

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

- 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

- 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@lfs458-node-1a0a:~$ kubectl scale deployment nginx --replicas=0
deployment.apps/nginx scaled
```

```
student@lfs458-node-1a0a:~$ kubectl get po
No resources found in default namespace.
```

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

```
student@lfs458-node-1a0a:~$ kubectl scale deployment nginx --replicas=2
deployment.apps/nginx scaled
```

```
student@lfs458-node-1a0a:~$ kubectl get po
```

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

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

```
student@lfs458-node-1a0a:~$ kubectl delete deployments nginx
deployment.apps "nginx" deleted
```

```
student@lfs458-node-1a0a:~$ kubectl delete ep nginx
endpoints "nginx" deleted
```

```
student@lfs458-node-1a0a:~$ kubectl delete svc nginx
service "nginx" deleted
```


Chapter 4

Kubernetes Architecture



4.1 Labs

Exercise 4.1: 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`, which we will name `hog`, to generate load. Verify you have a container running.

```
student@lfs458-node-1a0a:~$ kubectl create deployment hog --image vish/stress
deployment.apps/hog created
```

```
student@lfs458-node-1a0a:~$ kubectl get deployments
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
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@lfs458-node-1a0a:~$ kubectl describe deployment hog
```

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

```
student@lfs458-node-1a0a:~$ kubectl get deployment hog -o yaml
apiVersion: apps/v1
kind: Deployment
Metadata:

<output_omitted>

template:
  metadata:
    creationTimestamp: null
    labels:
      app: hog
  spec:
    containers:
      - image: vish/stress
        imagePullPolicy: Always
        name: stress
        resources: {}
        terminationMessagePath: /dev/termination-log
<output_omitted>
```

3. We will use the YAML output to create our own configuration file. The `--export` option can be useful to not include unique parameters. Again, the option has a deprecation message and may be removed in a future release.

```
student@lfs458-node-1a0a:~$ kubectl get deployment hog \
--export -o yaml > hog.yaml
```

4. If you did not use the `--export` option we will need to remove the status output, `creationTimestamp` and other settings, as we don't want to set unique generated parameters. We will also add in memory limits found below.

```
student@lfs458-node-1a0a:~$ vim hog.yaml
```

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@lfs458-node-1a0a:~$ kubectl replace -f hog.yaml
deployment.apps/hog replaced
```

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

```
student@lfs458-node-1a0a:~$ kubectl get deployment hog -o yaml
....
resources:
  limits:
    memory: 4Gi
  requests:
```

```

        memory: 2500Mi
        terminationMessagePath: /dev/termination-log
    ....

```

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

```

student@lfs458-node-1a0a:~$ kubectl get po
NAME                                READY   STATUS    RESTARTS   AGE
hog-64cbfcc7cf-lwq66               1/1     Running   0           2m

student@lfs458-node-1a0a:~$ kubectl logs hog-64cbfcc7cf-lwq66
I1102 16:16:42.638972      1 main.go:26] Allocating "0" memory, in
    "4Ki" chunks, with a 1ms sleep between allocations
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@lfs458-node-1a0a:~$ vim hog.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 be deployed to either.

```

student@lfs458-node-1a0a:~$ kubectl delete deployment hog
deployment.apps "hog" deleted

student@lfs458-node-1a0a:~$ kubectl create -f hog.yaml
deployment.apps/hog created

```



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@lfs458-node-1a0a:~$ kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
hog-1985182137-5bz2w	0/1	Error	1	5s

```
student@lfs458-node-1a0a:~$ kubectl logs hog-1985182137-5bz2w
```

```
panic: cannot parse '150mi': unable to parse quantity's suffix
```

```
goroutine 1 [running]:
panic(0x5ff9a0, 0xc820014cb0)
    /usr/local/go/src/runtime/panic.go:481 +0x3e6
k8s.io/kubernetes/pkg/api/resource.MustParse(0x7ffe460c0e69, 0x5, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0)
    /usr/local/google/home/vishnuk/go/src/k8s.io/kubernetes/pkg/api/resource/quantity.go:134 +0x287
main.main()
    /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@lfs458-node-1a0a:~$ kubectl get po
```

NAME	READY	STATUS	RESTARTS	AGE
hog-1603763060-x3vnn	1/1	Running	0	8s

```
student@lfs458-node-1a0a:~$ kubectl logs hog-1603763060-x3vnn
```

```
I0927 21:09:23.514921      1 main.go:26] Allocating "0" memory, in "4ki" chunks, with a 1ms sleep \
    between allocations
I0927 21:09:23.514984      1 main.go:39] Spawning a thread to consume CPU
I0927 21:09:23.514991      1 main.go:39] Spawning a thread to consume CPU
I0927 21:09:23.514997      1 main.go:29] Allocated "0" memory
```

✍ Exercise 4.2: 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@lfs458-node-1a0a:~$ kubectl create namespace low-usage-limit
```

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

```
student@lfs458-node-1a0a:~$ kubectl get namespace
```

NAME	STATUS	AGE
default	Active	1h
kube-node-lease	Active	1h
kube-public	Active	1h
kube-system	Active	1h
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@lfs458-node-1a0a:~$ vim low-resource-range.yaml
```

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@lfs458-node-1a0a:~$ kubectl --namespace=low-usage-limit \
  create -f low-resource-range.yaml
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@lfs458-node-1a0a:~$ kubectl get LimitRange
No resources found in default namespace.

student@lfs458-node-1a0a:~$ kubectl get LimitRange --all-namespaces
NAMESPACE      NAME                CREATED AT
low-usage-limit low-resource-range  2019-01-08T17:54:22
```

5. Create a new deployment in the namespace.

```
student@lfs458-node-1a0a:~$ kubectl -n low-usage-limit \
  create deployment limited-hog --image vish/stress
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@lfs458-node-1a0a:~$ kubectl get deployments --all-namespaces
```

NAMESPACE	NAME	READY	UP-TO-DATE	AVAILABLE	AGE
default	hog	1/1	1	1	7m57s
kube-system	calico-kube-controllers	1/1	1	1	2d10h
kube-system	coredns	2/2	2	2	2d10h
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@lfs458-node-1a0a:~$ kubectl -n low-usage-limit get pods
```

NAME	READY	STATUS	RESTARTS	AGE
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
<output_omitted>
spec:
  containers:
  - image: vish/stress
    imagePullPolicy: Always
    name: stress
    resources:
      limits:
        cpu: "1"
        memory: 500Mi
      requests:
        cpu: 500m
        memory: 100Mi
    terminationMessagePath: /dev/termination-log
<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@lfs458-node-1a0a:~$ cp hog.yaml hog2.yaml

student@lfs458-node-1a0a:~$ 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@lfs458-node-1a0a:~$ kubectl create -f hog2.yaml
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@lfs458-node-1a0a:~$ kubectl get deployments --all-namespaces
```

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

- 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.

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

- Delete the **hog** deployments to recover system resources.

```
student@lfs458-node-1a0a:~$ kubectl -n low-usage-limit delete deployment hog
deployment.apps "hog" deleted
```

```
student@lfs458-node-1a0a:~$ kubectl delete deployment hog
deployment.apps "hog" deleted
```

Exercise 4.3: Basic Node Maintenance

In this section we will cause some of our pods to be evicted from a node and rescheduled elsewhere. This could be part of basic maintenance or a rolling OS update.

- Create a deployment, then scale to create plenty of pods.

```
student@lfs458-node-1a0a:~$ kubectl create deployment maint --image=nginx
deployment.apps/maint created
```

```
student@lfs458-node-1a0a:~$ kubectl scale deployment maint --replicas=20
deployment.apps/maint scaled
```

- Use the terminal on the second node to get a count of the current docker containers. It should be something like 29, plus a line for status counted by **wc**. The main system should have something like 25 running, plus a line of status. Your numbers may be different.

```
student@lfs458-node-1a0a:~$ sudo docker ps | wc -l
26
```

```
student@lfs458-worker:~$ sudo docker ps | wc -l
30
```

- In order to complete maintenance we may need to move containers from a node and prevent new ones from deploying. One way to do this is to **drain**, or **cordons**, the node. Currently this will not affect DaemonSets, an object we will discuss in greater detail in the future. Begin by getting a list of nodes. Your node names will be different.

```
student@lfs458-node-1a0a:~$ kubectl get nodes

NAME                STATUS    ROLES    AGE     VERSION
lfs458-worker       Ready     <none>   4h      v1.16.1
lfs458-node-1a0a    Ready     master   4h      v1.16.1
```

- Modifying your second, worker node, update the node to **drain** the pods. Some resources may not drain, expect an error which we will work with next. Note the error includes aborting command which indicates the drain did not take place. Were you to check it would have the same number of containers running, but will show a new taint preventing the scheduler from assigning new pods.

```
student@lfs458-node-1a0a:~$ kubectl drain lfs458-worker
node/lfs458-worker cordoned
error: unable to drain node "lfs458-worker", aborting command...
```

```
There are pending nodes to be drained:
  lfs458-worker
error: DaemonSet-managed pods (use --ignore-daemonsets to ignore):
calico-node-vndn7, kube-proxy-rjjls
```

```
student@lfs458-node-1a0a:~$ kubectl describe node |grep -i taint
Taints:          <none>
Taints:          node.kubernetes.io/unschedulable:NoSchedule
```

5. As the error output suggests we can use the **–ignore-daemonsets** options to ignore containers which are not intended to move. We will find a new error when we use this command, near the end of the output. The node will continue to have the same number of pods and containers running.

```
student@lfs458-node-1a0a:~$ kubectl drain lfs458-worker --ignore-daemonsets
node/lfs458-worker already cordoned
WARNING: ignoring DaemonSet-managed Pods: kube-system/calico-node-dr279, kube-system/kube-proxy-qhc4f
evicting pod "maint-5bcc549596-b66fr"
evicting pod "maint-5bcc549596-2h5zg"
evicting pod "nginx-85ff79dd56-8h72c"
<output_omitted>
pod/maint-5bcc549596-xfbb6 evicted
pod/maint-5bcc549596-zmpdq evicted
pod/maint-5bcc549596-p2nnm evicted
node/lfs458-worker evicted
```

6. Run the command again. This time the output should both indicate the node has already been cordoned, then show the node has been drained. Not all pods will be gone as daemonsets will remain. Note the command is shown on two lines. You can omit the backslash and type the command on a single line.

```
student@lfs458-node-1a0a:~$ kubectl drain lfs458-worker \
--ignore-daemonsets --delete-local-data
node/lfs458-worker already cordoned
WARNING: ignoring DaemonSet-managed Pods: kube-system/calico-node-dr279, kube-system/kube-proxy-qhc4f
node/lfs458-worker drained
```

7. Were you to look on your second, worker node, you would see there should be fewer pods and containers than before. These pods can only be evicted via a special taint which we will discuss in the scheduling chapter.

```
student@lfs458-worker:~$ sudo docker ps | wc -l
5
```

8. Update the node taint such that the scheduler will use the node again. Verify that no nodes have moved over to the worker node as the scheduler only checks when a pod is deployed.

```
student@lfs458-node-1a0a:~$ kubectl uncordon lfs458-worker
node/lfs458-worker uncordoned

student@lfs458-node-1a0a:~$ kubectl describe node |grep -i taint
Taints:          <none>
Taints:          <none>

student@lfs458-worker:~$ sudo docker ps | wc -l
5

student@lfs458-node-1a0a:~$ sudo docker ps | wc -l
61
```

9. Clean up by deleting the maint deployment.

```
student@lfs458-worker:~$ kubectl delete deployment maint
deployment.apps "maint" 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@lfs458-node-1a0a:~$ less ~/.kube/config
```

```
<output_omitted>
```

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

```
student@lfs458-node-1a0a:~$ export client=$(grep client-cert ~/.kube/config |cut -d" " -f 6)
```

```
student@lfs458-node-1a0a:~$ echo $client
```

```
LSOtLS1CRUdJTiBDRVJUSUZJQ0FURSOtLS0tCk1JSUM4akNDQWRxZ0F3SUJ  
BZ01JRy9wbC9rWEpNdmd3RFFZSktvWklodmNOQVFFTEJRQXdGVEVUTUJFR0  
ExVUUKQXhNS2EzVmlaWEp1WlhSbGN6QWVGdzB4TnpFeU1UTXh0elEyTXpKY  
UZ3MHhPREV5TVRNeE56UTJNelJhTURReApGekFWQmdOVkJBb1REbk41YzNS  
<output_omitted>
```

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

```
student@lfs458-node-1a0a:~$ export key=$(grep client-key-data ~/.kube/config |cut -d " " -f 6)

student@lfs458-node-1a0a:~$ echo $key

<output_omitted>
```

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

```
student@lfs458-node-1a0a:~$ export auth=$(grep certificate-authority-data ~/.kube/config |cut -d " " -f 6)

student@lfs458-node-1a0a:~$ echo $auth

<output_omitted>
```

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

```
student@lfs458-node-1a0a:~$ echo $client | base64 -d - > ./client.pem

student@lfs458-node-1a0a:~$ echo $key | base64 -d - > ./client-key.pem

student@lfs458-node-1a0a:~$ 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@lfs458-node-1a0a:~$ kubectl config view |grep server

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@lfs458-node-1a0a:~$ 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 look for this file in the tarball output, it can save you some typing.

```
student@lfs458-node-1a0a:~$ vim curlpod.json
```

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

```

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

```

- 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@lfs458-node-1a0a:~$ 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>
  }
}

```

- Verify the new pod exists and shows a Running status.

```

student@lfs458-node-1a0a:~$ kubectl get pods

```

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

Exercise 5.2: Explore API Calls

- 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@lfs458-node-1a0a:~$ sudo apt-get install -y strace

```

```

student@lfs458-node-1a0a:~$ kubectl get endpoints

```

NAME	ENDPOINTS	AGE
kubernetes	10.128.0.3:6443	3h

- 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@lfs458-node-1a0a:~$ strace kubectl get endpoints
execve("/usr/bin/kubectl", ["kubectl", "get", "endpoints"], [/*...
....
openat(AT_FDCWD, "/home/student/.kube/cache/discovery/k8smaster_6443..
<output_omitted>

```

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

```

student@lfs458-node-1a0a:~$ cd /home/student/.kube/cache/discovery/

```

```

student@lfs458-node-1a0a:~/.kube/cache/discovery$ ls
k8smaster_6443

```

```

student@lfs458-node-1a0a:~/.kube/cache/discovery$ cd k8smaster_6443/

```

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

```
student@lfs458-node-1a0a: ~/.kube/cache/discovery/k8smaster_6443$ ls
admissionregistration.k8s.io  batch                node.k8s.io
apiextensions.k8s.io          certificates.k8s.io  policy
apiregistration.k8s.io       coordination.k8s.io rbac.authorization.k8s.io
apps                         crd.projectcalico.org scheduling.k8s.io
authentication.k8s.io        events.k8s.io       servergroups.json
authorization.k8s.io         extensions           storage.k8s.io
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@lfs458-node-1a0a: ./k8smaster_6443$ find .
.
./batch
./batch/v1
./batch/v1/serverresources.json
./batch/v1beta1
./batch/v1beta1/serverresources.json
./apiextensions.k8s.io
./apiextensions.k8s.io/v1
./apiextensions.k8s.io/v1/serverresources.json
./apiextensions.k8s.io/v1beta1
./apiextensions.k8s.io/v1beta1/serverresources.json
<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@lfs458-node-1a0a:.$ 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@lfs458-node-1a0a:.$ python3 -m json.tool v1/serverresources.json | less
```



serverresources.json

```
1  ....
2  {
3    "kind": "Endpoints",
4    "name": "endpoints",
```


JSON

```

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@lfs458-node-1a0a:.$ kubectl get ep
```

```

NAME           ENDPOINTS          AGE
kubernetes     10.128.0.3:6443    3h

```

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

```
student@lfs458-node-1a0a:.$ python3 -m json.tool v1/serverresources.json | grep kind
```

```

"kind": "APIResourceList",
"kind": "Binding",
"kind": "ComponentStatus",
"kind": "ConfigMap",
"kind": "Endpoints",
"kind": "Event",
<output_omitted>

```

10. Looking at another file we find nine more.

```
student@lfs458-node-1a0a:$ python3 -m json.tool \
  apps/v1/serverresources.json | grep kind
```

```

"kind": "APIResourceList",
"kind": "ControllerRevision",
"kind": "DaemonSet",
"kind": "DaemonSet",
"kind": "Deployment",
<output_omitted>

```

11. Delete the curlpod to recoup system resources.a

```
student@lfs458-node-1a0a:$ kubectl delete po curlpod
```

```
pod "curlpod" deleted
```

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

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@lfs458-node-1a0a:~$ kubectl config view
```

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: REDACTED
    server: https://k8smaster:6443
    name: kubernetes
<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@lfs458-node-1a0a:~$ kubectl get secrets --all-namespaces
```

```

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

```

```
student@lfs458-node-1a0a:~$ kubectl get secrets
```

```

NAME                                TYPE                                DATA  AGE
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@lfs458-node-1a0a:~$ kubectl describe secret default-token-jdqp7
```

```

Name:          default-token-jdqp7
Namespace:     default
Labels:        <none>
<output_omitted>
token:         eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJrdWJlcm5ldGVz
L3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWVjb3VudC9uYW1lc3Bh
Y2UiOiJkZWZhdWx0Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWVjb3VudC9zZW5yZXQubm
<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@lfs458-node-1a0a:~$ 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@lfs458-node-1a0a:~$ 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@lfs458-node-1a0a:~$ curl https://k8smaster:6443/api/v1 \
--header "Authorization: Bearer $token" -k
<output_omitted>

```

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

```

student@lfs458-node-1a0a:~$ curl \
https://k8smaster:6443/api/v1/namespaces \
--header "Authorization: Bearer $token" -k

```

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

- 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@lfs458-node-1a0a:~$ kubectl run -i -t busybox --image=busybox \
--restart=Never
```



Inside container

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

- Clean up by deleting the `busybox` container.

```
student@lfs458-node-1a0a:~$ kubectl delete pod busybox
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.

- 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@lfs458-node-1a0a:~$ kubectl proxy -h

Creates a proxy server or application-level gateway between localhost
and the Kubernetes API Server. It also allows serving static content
over specified HTTP path. All incoming data enters through one port
and gets forwarded to the remote kubernetes API Server port, except
for the path matching the static content path.
```

Examples:

```
# To proxy all of the kubernetes api and nothing else, use:

$ kubectl proxy --api-prefix=/
<output_omitted>
```

- Start the proxy while setting the API prefix, and put it in the background. You may need to use `enter` to view the prompt.

```
student@lfs458-node-1a0a:~$ kubectl proxy --api-prefix=/ &
[1] 22500
Starting to serve on 127.0.0.1:8001
```

- 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@lfs458-node-1a0a:~$ curl http://127.0.0.1:8001/api/
```

```
<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@lfs458-node-1a0a:~$ curl http://127.0.0.1:8001/api/v1/namespaces
```

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

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@lfs458-node-1a0a:~$ vim job.yaml
```

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@lfs458-node-1a0a:~$ kubectl create -f job.yaml
```

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

```
student@lfs458-node-1a0a:~$ kubectl get job
```

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

```
student@lfs458-node-1a0a:~$ kubectl describe jobs.batch sleepy
```

```

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

```

```
student@lfs458-node-1a0a:~$ kubectl get job
```

```

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

```

- 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@lfs458-node-1a0a:~$ kubectl get jobs.batch sleepy -o yaml
```

```

<output_omitted>
uid: c2c3a80d-d0fc-11e8-947a-42010a800002
spec:
  backoffLimit: 6
  completions: 1
  parallelism: 1
  selector:
    matchLabels:
<output_omitted>

```

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

```

student@lfs458-node-1a0a:~$ kubectl delete jobs.batch sleepy
job.batch "sleepy" deleted

```

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

```
student@lfs458-node-1a0a:~$ vim job.yaml
```

YAML

job.yaml

```

1 <output_omitted>
2 metadata:
3   name: sleepy
4 spec:
5   completions: 5  #<--Add this line
6   template:
7     spec:
8       containers:
9 <output_omitted>

```

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

```

student@lfs458-node-1a0a:~$ kubectl create -f job.yaml
job.batch/sleepy created

```

```
student@lfs458-node-1a0a:~$ kubectl get jobs.batch
```

NAME	COMPLETIONS	DURATION	AGE
sleepy	0/5	5s	5s

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

```
student@lfs458-node-1a0a:~$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
sleepy-z5tnh	0/1	Completed	0	8s
sleepy-zd692	1/1	Running	0	3s

<output_omitted>

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

```
student@lfs458-node-1a0a:~$ kubectl get jobs
```

NAME	COMPLETIONS	DURATION	AGE
sleepy	5/5	26s	10m

```
student@lfs458-node-1a0a:~$ kubectl delete jobs.batch sleepy
```

```
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@lfs458-node-1a0a:~$ vim job.yaml
```

YAML

job.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@lfs458-node-1a0a:~$ kubectl create -f job.yaml
```

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

```
student@lfs458-node-1a0a:~$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
sleepy-8xwpc	1/1	Running	0	5s
sleepy-xjqnf	1/1	Running	0	5s

<output_omitted>

```
student@lfs458-node-1a0a:~$ kubectl get jobs
```

NAME	COMPLETIONS	DURATION	AGE
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@lfs458-node-1a0a:~$ 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@lfs458-node-1a0a:~$ kubectl delete jobs.batch sleepy
job.batch "sleepy" deleted

```

```

student@lfs458-node-1a0a:~$ kubectl create -f job.yaml
job.batch/sleepy created

```

```

student@lfs458-node-1a0a:~$ kubectl get jobs

```

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

```

student@lfs458-node-1a0a:~$ kubectl get jobs

```

NAME	COMPLETIONS	DURATION	AGE
sleepy	3/5	16s	16s

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

```

student@lfs458-node-1a0a:~$ kubectl get job sleepy -o yaml
<output_omitted>
status:
  conditions:
  - lastProbeTime: 2018-10-16T05:45:14Z
    lastTransitionTime: 2018-10-16T05:45:14Z
    message: Job was active longer than specified deadline
    reason: DeadlineExceeded
    status: "True"
    type: Failed
  failed: 2
  startTime: 2018-10-16T05:44:59Z
  succeeded: 3

```

14. Delete the job.

```

student@lfs458-node-1a0a:~$ kubectl delete jobs.batch sleepy
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@lfs458-node-1a0a:~$ 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@lfs458-node-1a0a:~$ vim cronjob.yaml
```

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: "*/* * * * *"      #<-- 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@lfs458-node-1a0a:~$ kubectl create -f cronjob.yaml
```

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

```
student@lfs458-node-1a0a:~$ kubectl get cronjobs.batch
```

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

```
student@lfs458-node-1a0a:~$ kubectl get jobs.batch
```

```
No resources found.
```

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

```
student@lfs458-node-1a0a:~$ kubectl get cronjobs.batch
```

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

```
student@lfs458-node-1a0a:~$ kubectl get jobs.batch
```

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

```
student@lfs458-node-1a0a:~$ kubectl get jobs.batch
```

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

- 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@lfs458-node-1a0a:~$ vim cronjob.yaml
```

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          ....
```

- 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@lfs458-node-1a0a:~$ kubectl delete cronjobs.batch sleepy
cronjob.batch "sleepy" deleted
```

```
student@lfs458-node-1a0a:~$ kubectl create -f cronjob.yaml
cronjob.batch/sleepy created
```

```
student@lfs458-node-1a0a:~$ kubectl get jobs
```

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

```
student@lfs458-node-1a0a:~$ kubectl get cronjobs.batch
```

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

```
student@lfs458-node-1a0a:~$ kubectl get jobs
```

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

```
student@lfs458-node-1a0a:~$ kubectl get jobs
```

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

```
student@lfs458-node-1a0a:~$ kubectl get cronjobs.batch
```

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

- Clean up by deleting the CronJob.

```
student@lfs458-node-1a0a:~$ kubectl delete cronjobs.batch sleepy
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@lfs458-node-1a0a:~$ kubectl get rs
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@lfs458-node-1a0a:~$ vim rs.yaml
```



rs.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:1.11.1
18         ports:
19         - containerPort: 80

```

3. Create the ReplicaSet:

```

student@lfs458-node-1a0a:~$ kubectl create -f rs.yaml
replicaset.apps/rs-one created

```

4. View the newly created ReplicaSet:

```

student@lfs458-node-1a0a:~$ kubectl describe rs rs-one
Name:                rs-one
Namespace:            default
Selector:             system=ReplicaOne
Labels:               system=ReplicaOne
Annotations:          <none>
Replicas:             2 current / 2 desired
Pods Status:          2 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:             system=ReplicaOne
  Containers:
    nginx:
      Image:           nginx:1.11.1
      Port:            80/TCP
      Host Port:       0/TCP
      Environment:    <none>
      Mounts:          <none>
      Volumes:         <none>
  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@lfs458-node-1a0a:~$ kubectl get pods
NAME                READY    STATUS    RESTARTS   AGE
rs-one-2p9x4        1/1      Running   0           5m4s
rs-one-3c6pb        1/1      Running   0           5m4s

```

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

```

student@lfs458-node-1a0a:~$ kubectl delete rs rs-one --cascade=false
replicaset.apps "rs-one" deleted

```

7. View the ReplicaSet and Pods again:

```
student@lfs458-node-1a0a:~$ kubectl describe rs rs-one
Error from server (NotFound): replicaset.apps "rs-one" not found

student@lfs458-node-1a0a:~$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
rs-one-2p9x4	1/1	Running	0	7m
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@lfs458-node-1a0a:~$ kubectl create -f rs.yaml
replicaset.apps/rs-one created
```

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

```
student@lfs458-node-1a0a:~$ kubectl get rs
```

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

```
student@lfs458-node-1a0a:~$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
rs-one-2p9x4	1/1	Running	0	8m
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@lfs458-node-1a0a:~$ kubectl edit po rs-one-3c6pb

....
  labels:
    system: IsolatedPod  #<-- Change from ReplicaOne
    name: rs-one-3c6pb
....
```

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

```
student@lfs458-node-1a0a:~$ kubectl get rs
```

NAME	DESIRED	CURRENT	READY	AGE
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@lfs458-node-1a0a:~$ kubectl get po -L system
```

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

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

```
student@lfs458-node-1a0a:~$ kubectl delete rs rs-one
replicaset.apps "rs-one" deleted
```

```
student@lfs458-node-1a0a:~$ kubectl get po
```

NAME	READY	STATUS	RESTARTS	AGE
rs-one-3c6pb	1/1	Running	0	14m
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@lfs458-node-1a0a:~$ kubectl get rs
```

No resources found in default namespaces.

```
student@lfs458-node-1a0a:~$ kubectl get po
```

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

15. Delete the remaining Pod using the label.

```
student@lfs458-node-1a0a:~$ kubectl delete po -l system=IsolatedPod
```

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@lfs458-node-1a0a:~$ cp rs.yaml ds.yaml
```

```
student@lfs458-node-1a0a:~$ vim ds.yaml
```

YAML

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@lfs458-node-1a0a:~$ kubectl create -f ds.yaml
```



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

```
student@lfs458-node-1a0a:~$ kubectl get ds
```

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

```
student@lfs458-node-1a0a:~$ kubectl get po
```

NAME	READY	STATUS	RESTARTS	AGE
ds-one-b1dcv	1/1	Running	0	2m
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@lfs458-node-1a0a:~$ kubectl describe po ds-one-b1dcv | grep Image:
```

```
Image: nginx:1.11.1
```

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@lfs458-node-1a0a:~$ 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@lfs458-node-1a0a:~$ 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 `1.12.1-alpine`.

```
student@lfs458-node-1a0a:~$ kubectl set image ds ds-one nginx=nginx:1.12.1-alpine
daemonset.apps/ds-one image updated
```

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

```
student@lfs458-node-1a0a:~$ kubectl describe po ds-one-b1dcv |grep Image:
Image:                nginx:1.11.1
```

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

```
student@lfs458-node-1a0a:~$ kubectl delete po ds-one-b1dcv
pod "ds-one-b1dcv" deleted
```

```
student@lfs458-node-1a0a:~$ kubectl get po
```

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

```
student@lfs458-node-1a0a:~$ kubectl describe po ds-one-xc86w |grep Image:
Image:                nginx:1.12.1-alpine
```

6. View the image running on the older Pod. It should still show version 1.11.1.

```
student@lfs458-node-1a0a:~$ kubectl describe po ds-one-z31r4 |grep Image:
Image:                nginx:1.11.1
```

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@lfs458-node-1a0a:~$ 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@lfs458-node-1a0a:~$ kubectl rollout history ds ds-one --revision=1
```

```
daemonsets "ds-one" with revision #1
Pod Template:
  Labels:            system=DaemonSetOne
  Containers:
    nginx:
      Image:          nginx:1.11.1
      Port:           80/TCP
      Environment:    <none>
      Mounts:         <none>
      Volumes:        <none>
```

```
student@lfs458-node-1a0a:~$ kubectl rollout history ds ds-one --revision=2
```

```
....
Image:          nginx:1.12.1-alpine
.....
```

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@lfs458-node-1a0a:~$ kubectl rollout undo ds ds-one --to-revision=1
daemonset.apps/ds-one rolled back
```

```
student@lfs458-node-1a0a:~$ kubectl describe po ds-one-xc86w |grep Image:
```

```
Image:          nginx:1.12.1-alpine
```

10. Delete the Pod, wait for the replacement to spawn then check the image version again.

```
student@lfs458-node-1a0a:~$ kubectl delete po ds-one-xc86w
```

```
pod "ds-one-xc86w" deleted
```

```
student@lfs458-node-1a0a:~$ kubectl get po
```

NAME	READY	STATUS	RESTARTS	AGE
ds-one-qc72k	1/1	Running	0	10s
ds-one-xc86w	0/1	Terminating	0	12m
ds-one-z31r4	1/1	Running	0	28m

```
student@lfs458-node-1a0a:~$ kubectl describe po ds-one-qc72k |grep Image:
```

```
Image:          nginx:1.11.1
```

11. View the details of the DaemonSet. The Image should be v1.11.1 in the output.

```
student@lfs458-node-1a0a:~$ kubectl describe ds |grep Image:
```

```
Image:          nginx:1.11.1
```

12. View the current configuration for the DaemonSet in YAML output. Look for the update strategy near the end of the output.

```
student@lfs458-node-1a0a:~$ 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. You will get a deprecation warning.

```
student@lfs458-node-1a0a:~$ kubectl get ds ds-one -o yaml --export > ds2.yaml
```

```
Flag --export has been deprecated, This flag is deprecated
and will be removed in future.
```

14. Edit the file. Change the name, around line eight and the update strategy around line 38, back to the default RollingUpdate.

```
student@lfs458-node-1a0a:~$ vim ds2.yaml
```

```
....
  name: ds-two
....
  type: RollingUpdate
```

15. Create the new DaemonSet and verify the **nginx** version in the new pods.

```
student@lfs458-node-1a0a:~$ kubectl create -f ds2.yaml
```

```
daemonset.apps/ds-two created
```

```
student@lfs458-node-1a0a:~$ kubectl get po
```

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@lfs458-node-1a0a:~$ kubectl describe po ds-two-10khc |grep Image:
Image:                nginx:1.11.1
```

16. Edit the configuration file and set the image to a newer version such as 1.12.1-alpine.

```
student@lfs458-node-1a0a:~$ kubectl edit ds ds-two
```

```
....
- image: nginx:1.12.1-alpine
.....
```

17. View the age of the DaemonSets. It should be around ten minutes old, depending on how fast you type.

```
student@lfs458-node-1a0a:~$ 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@lfs458-node-1a0a:~$ kubectl get po
```

NAME	READY	STATUS	RESTARTS	AGE
ds-one-qc72k	1/1	Running	0	36m
ds-one-z31r4	1/1	Running	0	1h
ds-two-2p8vz	1/1	Running	0	34s
ds-two-8lx7k	1/1	Running	0	32s

19. Verify the Pods are using the new version of the software.

```
student@lfs458-node-1a0a:~$ kubectl describe po ds-two-8lx7k |grep Image:
Image:                nginx:1.12.1-alpine
```

20. View the rollout status and the history of the DaemonSets.

```
student@lfs458-node-1a0a:~$ kubectl rollout status ds ds-two
daemon set "ds-two" successfully rolled out

student@lfs458-node-1a0a:~$ kubectl rollout history ds ds-two
daemonsets "ds-two"
REVISION    CHANGE-CAUSE
1           <none>
2           <none>
```

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@lfs458-node-1a0a:~$ kubectl rollout history ds ds-two --revision=2
...
Image:                nginx:1.12.1-alpine
```

22. Clean up the system by removing the DaemonSets.

```
student@lfs458-node-1a0a:~$ kubectl delete ds ds-one ds-two
daemonset.apps "ds-one" deleted
daemonset.apps "ds-two" deleted
```

Chapter 8

Services



8.1 Labs

Exercise 8.1: 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. We will use the `v1beta` version of the API. 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@lfs458-node-1a0a:~$ vim nginx-one.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:1.11.1
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@lfs458-node-1a0a:~$ kubectl get nodes --show-labels
<output_omitted>

```

3. Run the following command and look for the errors. Assuming there is no typo, you should have gotten an error about the accounting namespace.

```

student@lfs458-node-1a0a:~$ kubectl create -f nginx-one.yaml
Error from server (NotFound): error when creating
"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@lfs458-node-1a0a:~$ kubectl create ns accounting
namespace/accounting" created

```

```
student@lfs458-node-1a0a:~$ kubectl create -f nginx-one.yaml
deployment.apps/nginx-one created
```

5. View the status of the new pods. Note they do not show a Running status.

```
student@lfs458-node-1a0a:~$ kubectl -n accounting get pods
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-one-74dd9d578d-fcpmv	0/1	Pending	0	4m
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@lfs458-node-1a0a:~$ kubectl -n accounting describe pod \
    nginx-one-74dd9d578d-fcpmv
```

Name: nginx-one-74dd9d578d-fcpmv
 Namespace: accounting
 Node: <none>

<output_omitted>

Events:

Type	Reason	Age	From
Warning	FailedScheduling	<unknown>	default-scheduler	

0/2 nodes are available: 2 node(s) didn't match node selector.

7. Label the secondary node. Verify the labels.

```
student@lfs458-node-1a0a:~$ kubectl label node lfs458-worker \
    system=secondOne
node/lfs458-worker labeled
```

```
student@lfs458-node-1a0a:~$ kubectl get nodes --show-labels
```

NAME	STATUS	ROLES	AGE	VERSION	LABELS
lfs458-node-1a0a	Ready	master	1d1h	v1.12.1	\ beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/ hostname=lfs458-node-1a0a,node-role.kubernetes.io/master=
lfs458-worker	Ready	<none>	1d1h	v1.12.1	\ beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/ hostname=lfs458-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@lfs458-node-1a0a:~$ kubectl -n accounting get pods
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-one-74dd9d578d-fcpmv	1/1	Running	0	10m
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@lfs458-node-1a0a:~$ kubectl get pods -l system=secondary --all-namespaces
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
accounting	nginx-one-74dd9d578d-fcpmv	1/1	Running	0	20m
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@lfs458-node-1a0a:~$ kubectl -n accounting expose deployment nginx-one
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
```

NAME	ENDPOINTS	AGE
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@lfs458-node-1a0a:~$ curl 192.168.1.72:8080
curl: (7) Failed to connect to 192.168.1.72 port 8080: Connection refused

student@lfs458-node-1a0a:~$ curl 192.168.1.72:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<output_omitted>
```

13. Delete the deployment. Edit the YAML file to expose port 80 and create the deployment again.

```
student@lfs458-node-1a0a:~$ kubectl -n accounting delete deploy nginx-one
deployment.apps "nginx-one" deleted

student@lfs458-node-1a0a:~$ vim nginx-one.yaml
```

YAML

nginx-one.yaml

```
1 ....
2     ports:
3     - containerPort: 8080      #<-- Edit this line
4       protocol: TCP
5     ....
```

```
student@lfs458-node-1a0a:~$ kubectl create -f nginx-one.yaml
deployment.apps/nginx-one created
```

✍ Exercise 8.2: 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@lfs458-node-1a0a:~$ kubectl -n accounting expose deployment \
    nginx-one --type=NodePort --name=service-lab
service/service-lab exposed
```


2. View the details of the services in the accounting namespace. We are looking for the autogenerated port.

```
student@lfs458-node-1a0a:~$ kubectl -n accounting describe services
....
NodePort:                <unset> 32103/TCP
....
```

3. Locate the exterior facing hostname or IP address of the cluster. As we are using GCP nodes, which we access via a FloatingIP, we will first check the internal only public IP address. Look for the Kubernetes master URL.

```
student@lfs458-node-1a0a:~$ kubectl cluster-info
Kubernetes master is running at https://k8smaster:6443
KubeDNS is running at https://k8smaster:6443/api/v1/namespaces/
kube-system/services/kube-dns/proxy
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@lfs458-node-1a0a:~$ curl http://k8smaster:32103
<!DOCTYPE html>
<html>
<head>
<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 8.3: Use Labels to Manage Resources

1. Try to delete all Pods with the `system=secondary` label, in all namespaces.

```
student@lfs458-node-1a0a:~$ kubectl delete pods -l system=secondary \
--all-namespaces
pod "nginx-one-74dd9d578d-fcpmv" deleted
pod "nginx-one-74dd9d578d-sts51" deleted
```

2. View the Pods again. New versions of the Pods should be running as the controller responsible for them continues.

```
student@lfs458-node-1a0a:~$ kubectl -n accounting get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-one-74dd9d578d-ddt5r         1/1     Running   0           1m
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@lfs458-node-1a0a:~$ kubectl -n accounting get deploy --show-labels
NAME      READY  UP-TO-DATE  AVAILABLE  AGE  LABELS
nginx-one  2/2    2           2           10m  system=secondary
```

4. Delete the deployment using its label.

```
student@lfs458-node-1a0a:~$ kubectl -n accounting delete deploy \
-l system=secondary
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@lfs458-node-1a0a:~$ kubectl label node lfs458-worker system-
node/lfs458-worker labeled
```


Chapter 9

Volumes and Data



9.1 Labs

Exercise 9.1: 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@lfs458-node-1a0a:~$ mkdir primary
```

```
student@lfs458-node-1a0a:~$ echo c > primary/cyan
```

```
student@lfs458-node-1a0a:~$ echo m > primary/magenta
```

```
student@lfs458-node-1a0a:~$ echo y > primary/yellow
student@lfs458-node-1a0a:~$ echo k > primary/black
student@lfs458-node-1a0a:~$ echo "known as key" >> primary/black
student@lfs458-node-1a0a:~$ 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@lfs458-node-1a0a:~$ kubectl create configmap colors \
  --from-literal=text=black \
  --from-file=./favorite \
  --from-file=./primary/
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@lfs458-node-1a0a:~$ kubectl get configmap colors

NAME      DATA      AGE
colors    6          30s

student@lfs458-node-1a0a:~$ kubectl get configmap colors -o yaml
apiVersion: v1
data:
  black: |
    k
    known as key
  cyan: |
    c
  favorite: |
    blue
  magenta: |
    m
  text: black
  yellow: |
    y
kind: ConfigMap
<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@lfs458-node-1a0a:~$ vim simpleshell.yaml
```

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@lfs458-node-1a0a:~$ kubectl create -f simpleshell.yaml
pod/shell-demo created

student@lfs458-node-1a0a:~$ kubectl exec shell-demo -- /bin/bash -c 'echo $ilike'
blue

student@lfs458-node-1a0a:~$ kubectl delete pod shell-demo
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@lfs458-node-1a0a:~$ 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@lfs458-node-1a0a:~$ kubectl create -f simpleshell.yaml
pod/shell-demo created

student@lfs458-node-1a0a:~$ kubectl exec shell-demo -- /bin/bash -c 'env'
black=k
known as key

KUBERNETES_SERVICE_PORT_HTTPS=443
cyan=c
<output_omitted>

student@lfs458-node-1a0a:~$ kubectl delete pod shell-demo
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@lfs458-node-1a0a:~$ vim car-map.yaml
```

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@lfs458-node-1a0a:~$ kubectl create -f car-map.yaml
configmap/fast-car created

```

```

student@lfs458-node-1a0a:~$ kubectl get configmap fast-car -o yaml

```

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@lfs458-node-1a0a:~$ vim simpleshell.yaml

```

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@lfs458-node-1a0a:~$ kubectl create -f simpleshell.yaml
pod "shell-demo" created

```

```
student@lfs458-node-1a0a:~$ kubectl exec shell-demo -- /bin/bash -c 'df -ha |grep car'
/dev/root          9.6G  3.2G   6.4G   34% /etc/cars

student@lfs458-node-1a0a:~$ kubectl exec shell-demo -- /bin/bash -c 'cat /etc/cars/car.trim'
Shelby #<-- Then your prompt
```

11. Delete the Pod and ConfigMaps we were using.

```
student@lfs458-node-1a0a:~$ kubectl delete pods shell-demo
pod "shell-demo" deleted

student@lfs458-node-1a0a:~$ kubectl delete configmap fast-car colors
configmap "fast-car" deleted
configmap "colors" deleted
```

Exercise 9.2: 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@lfs458-node-1a0a:~$ sudo apt-get update && sudo \
    apt-get install -y nfs-kernel-server
<output_omitted>
```

2. Make and populate a directory to be shared. Also give it similar permissions to `/tmp/`

```
student@lfs458-node-1a0a:~$ sudo mkdir /opt/sfw

student@lfs458-node-1a0a:~$ sudo chmod 1777 /opt/sfw/

student@lfs458-node-1a0a:~$ 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@lfs458-node-1a0a:~$ sudo vim /etc/exports
/opt/sfw/ *(rw,sync,no_root_squash,subtree_check)
```

4. Cause `/etc/exports` to be re-read:

```
student@lfs458-node-1a0a:~$ sudo exportfs -ra
```

5. Test by mounting the resource from your **second** node.

```
student@lfs458-worker:~$ sudo apt-get -y install nfs-common
<output_omitted>

student@lfs458-worker:~$ showmount -e k8smaster
Export list for k8smaster:
/opt/sfw *

student@lfs458-worker:~$ sudo mount k8smaster:/opt/sfw /mnt

student@lfs458-worker:~$ ls -l /mnt
```

```
total 4
-rw-r--r-- 1 root root 9 Sep 28 17:55 hello.txt
```

- 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@lfs458-node-1a0a:~$ vim PVol.yaml
```

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
```

- Create the persistent volume, then verify its creation.

```
student@lfs458-node-1a0a:~$ kubectl create -f PVol.yaml
persistentvolume/pvvol-1 created
```

```
student@lfs458-node-1a0a:~$ kubectl get pv
```

NAME	CAPACITY	ACCESSMODES	RECLAIMPOLICY	STATUS
CLAIM	STORAGECLASS	REASON	AGE	
pvvol-1	1Gi	RWX	Retain	Available 4s

✍ Exercise 9.3: Creating a Persistent Volume Claim (PVC)

Before Pods can take advantage of the new PV we need to create a **Persistent Volume Claim (PVC)**.

- Begin by determining if any currently exist.

```
student@lfs458-node-1a0a:~$ kubectl get pvc
No resources found in default namespace.
```

- Create a YAML file for the new pvc.

```
student@lfs458-node-1a0a:~$ 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@lfs458-node-1a0a:~$ kubectl create -f pvc.yaml
persistentvolumeclaim/pvc-one created

```

```

student@lfs458-node-1a0a:~$ kubectl get pvc

```

NAME	STATUS	VOLUME	CAPACITY	ACCESSMODES	STORAGECLASS	AGE
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@lfs458-node-1a0a:~$ kubectl get pv

```

NAME	CAPACITY	ACCESSMODES	RECLAIMPOLICY	STATUS	CLAIM
pvvol-1	1Gi	RWX	Retain	Bound	default/pvc-one

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@lfs458-node-1a0a:~$ cp first.yaml nfs-pod.yaml

```

```

student@lfs458-node-1a0a:~$ vim nfs-pod.yaml

```



nfs-pod.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
10   namespace: default
11   resourceVersion: "1411"
12  spec:
13    replicas: 1
14    selector:
15      matchLabels:
16        run: nginx
17    strategy:
18      rollingUpdate:
19        maxSurge: 1

```



```

20     maxUnavailable: 1
21     type: RollingUpdate
22   template:
23     metadata:
24       creationTimestamp: null
25       labels:
26         run: nginx
27     spec:
28       containers:
29       - image: nginx
30         imagePullPolicy: Always
31         name: nginx
32         volumeMounts:
33         - name: nfs-vol
34           mountPath: /opt
35       ports:
36       - containerPort: 80
37         protocol: TCP
38       resources: {}
39       terminationMessagePath: /dev/termination-log
40       terminationMessagePolicy: File
41     volumes:                                     #<<-- These four lines
42     - name: nfs-vol
43       persistentVolumeClaim:
44         claimName: pvc-one
45     dnsPolicy: ClusterFirst
46     restartPolicy: Always
47     schedulerName: default-scheduler
48     securityContext: {}
49     terminationGracePeriodSeconds: 30

```

6. Create the pod using the newly edited file.

```

student@lfs458-node-1a0a:~$ kubectl create -f nfs-pod.yaml
deployment.apps/nginx-nfs created

```

7. Look at the details of the pod. You may see the daemonset pods running as well.

```

student@lfs458-node-1a0a:~$ kubectl get pods

```

NAME	READY	STATUS	RESTARTS	AGE
nginx-nfs-1054709768-s8g28	1/1	Running	0	3m

```

student@lfs458-node-1a0a:~$ kubectl describe pod nginx-nfs-1054709768-s8g28
Name:          nginx-nfs-1054709768-s8g28
Namespace:     default
Priority:       0
Node:          lfs458-worker/10.128.0.5
<output_omitted>

Mounts:
  /opt from nfs-vol (rw)
<output_omitted>

Volumes:
  nfs-vol:
    Type:      PersistentVolumeClaim (a reference to a PersistentV...

```

```

    ClaimName:      pvc-one
    ReadOnly:       false
    <output_omitted>

```

8. View the status of the PVC. It should show as bound.

```

student@lfs458-node-1a0a:~$ kubectl get pvc
NAME      STATUS VOLUME  CAPACITY ACCESS MODES  STORAGECLASS  AGE
pvc-one   Bound  pvvol-1  1Gi      RWX                


```

Exercise 9.4: 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@lfs458-node-1a0a:~$ kubectl delete deploy nginx-nfs
deployment.apps "nginx-nfs" deleted

student@lfs458-node-1a0a:~$ kubectl delete pvc pvc-one
persistentvolumeclaim "pvc-one" deleted

student@lfs458-node-1a0a:~$ kubectl delete pv pvvol-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@lfs458-node-1a0a:~$ 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@lfs458-node-1a0a:~$ kubectl create namespace small
namespace/small created

student@lfs458-node-1a0a:~$ kubectl describe ns small

Name:      small
Labels:     <none>
Annotations: <none>
Status:     Active

No resource quota.

No resource limits.

```

4. Create a new pv and pvc in the small namespace.

```
student@lfs458-node-1a0a:~$ kubectl -n small create -f PVol.yaml
persistentvolume/pvvol-1 created

student@lfs458-node-1a0a:~$ kubectl -n small create -f pvc.yaml
persistentvolumeclaim/pvc-one created
```

5. Create the new resource quota, placing this object into the small namespace.

```
student@lfs458-node-1a0a:~$ kubectl -n small create -f storage-quota.yaml
resourcequota/storagequota created
```

6. Verify the small namespace has quotas. Compare the output to the same command above.

```
student@lfs458-node-1a0a:~$ kubectl describe ns small
```

```
Name:      small
Labels:    <none>
Annotations: <none>
Status:    Active
```

Resource Quotas

```
Name:      storagequota
Resource   Used   Hard
-----
persistentvolumeclaims 1     10
requests.storage      200Mi 500Mi
```

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@lfs458-node-1a0a:~$ vim nfs-pod.yaml
```

8. Create the container again.

```
student@lfs458-node-1a0a:~$ kubectl -n small create -f nfs-pod.yaml
deployment.apps/nginx-nfs created
```

9. Determine if the deployment has a running pod.

```
student@lfs458-node-1a0a:~$ kubectl -n small get deploy
```

```
NAME      READY  UP-TO-DATE  AVAILABLE  AGE
nginx-nfs  1/1    1           1          43s
```

```
student@lfs458-node-1a0a:~$ kubectl -n small describe deploy nginx-nfs
```

<output_omitted>

10. Look to see if the pods are ready.

```
student@lfs458-node-1a0a:~$ kubectl -n small get pod
```

```
NAME                                READY  STATUS    RESTARTS  AGE
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@lfs458-node-1a0a:~$ kubectl -n small describe pod \
    nginx-nfs-2854978848-g3khf

Name:          nginx-nfs-2854978848-g3khf
Namespace:     small
<output_omitted>

Mounts:
  /opt from nfs-vol (rw)
<output_omitted>
```

12. View the quota usage of the namespace

```
student@lfs458-node-1a0a:~$ kubectl describe ns small

<output_omitted>

Resource Quotas
Name:          storagequota
Resource       Used   Hard
-----
persistentvolumeclaims 1    10
requests.storage 200Mi 500Mi

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@lfs458-node-1a0a:~$ sudo dd if=/dev/zero of=/opt/sfw/bigfile bs=1M count=300

300+0 records in
300+0 records out
314572800 bytes (315 MB, 300 MiB) copied, 0.196794 s, 1.6 GB/s

student@lfs458-node-1a0a:~$ kubectl describe ns small

<output_omitted>
Resource Quotas
Name:          storagequota
Resource       Used   Hard
-----
persistentvolumeclaims 1    10
requests.storage 200Mi 500Mi
<output_omitted>

student@lfs458-node-1a0a:~$ du -h /opt/

301M    /opt/sfw
41M     /opt/cni/bin
41M     /opt/cni
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@lfs458-node-1a0a:~$ kubectl -n small get deploy

NAME          READY   UP-TO-DATE   AVAILABLE   AGE
nginx-nfs     1       1             1           11m

student@lfs458-node-1a0a:~$ kubectl -n small delete deploy nginx-nfs

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@lfs458-node-1a0a:~$ kubectl describe ns small
```

```
<output_omitted>
```

```
Resource Quotas
```

Name:	storagequota	
Resource	Used	Hard
-----	---	---
persistentvolumeclaims	1	10
requests.storage	200Mi	500Mi

16. Remove the pvc then view the pv it was using. Note the RECLAIM POLICY and STATUS.

```
student@lfs458-node-1a0a:~$ kubectl -n small get pvc
```

NAME	STATUS	VOLUME	CAPACITY	ACCESSMODES	STORAGECLASS	AGE
pvc-one	Bound	pvvol-1	1Gi	RWX		19m

```
student@lfs458-node-1a0a:~$ kubectl -n small delete pvc pvc-one
```

```
persistentvolumeclaim "pvc-one" deleted
```

```
student@lfs458-node-1a0a:~$ kubectl -n small get pv
```

NAME	CAPACITY	ACCESSMODES	RECLAIMPOLICY	STATUS	CLAIM
STORAGECLASS	REASON	AGE			
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@lfs458-node-1a0a:~$ kubectl get pv/pvvol-1 -o yaml
```

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@lfs458-node-1a0a:~$ kubectl delete pv/pvvol-1
```

```
persistentvolume "pvvol-1" deleted
```

```
student@lfs458-node-1a0a:~$ grep Retain PVol.yaml
```

```
  persistentVolumeReclaimPolicy: Retain
```

```
student@lfs458-node-1a0a:~$ kubectl create -f PVol.yaml
```

```
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@lfs458-node-1a0a:~$ kubectl patch pv pvvol-1 -p \
'{"spec":{"persistentVolumeReclaimPolicy":"Delete"}}'
```

persistentvolume/pvvol-1 patched

```
student@lfs458-node-1a0a:~$ kubectl get pv/pvvol-1
```

NAME	CAPACITY	ACCESSMODES	RECLAIMPOLICY	STATUS	CLAIM
STORAGECLASS	REASON	AGE			
pvvol-1	1Gi	RWX	Delete	Available	2m

20. View the current quota settings.

```
student@lfs458-node-1a0a:~$ kubectl describe ns small
```

```
....
requests.storage      0      500Mi
```

21. Create the pvc again. Even with no pods running, note the resource usage.

```
student@lfs458-node-1a0a:~$ kubectl -n small create -f pvc.yaml
```

persistentvolumeclaim/pvc-one created

```
student@lfs458-node-1a0a:~$ kubectl describe ns small
```

```
....
requests.storage      200Mi      500Mi
```

22. Remove the existing quota from the namespace.

```
student@lfs458-node-1a0a:~$ kubectl -n small get resourcequota
```

NAME	CREATED AT
storagequota	2019-11-25T04:10:02Z

```
student@lfs458-node-1a0a:~$ kubectl -n small delete resourcequota storagequota
```

resourcequota "storagequota" deleted

23. Edit the storagequota.yaml file and lower the capacity to 100Mi.

```
student@lfs458-node-1a0a:~$ vim storage-quota.yaml
```

YAML

```
1 ....
2 requests.storage: "100Mi"
```

24. Create and verify the new storage quota. Note the hard limit has already been exceeded.

```
student@lfs458-node-1a0a:~$ kubectl -n small create -f storage-quota.yaml
```

resourcequota/storagequota created

```
student@lfs458-node-1a0a:~$ kubectl describe ns small
```

```
....
persistentvolumeclaims      1      10
requests.storage      200Mi      100Mi
```

No resource limits.

25. Create the deployment again. View the deployment. Note there are no errors seen.

```
student@lfs458-node-1a0a:~$ kubectl -n small create -f nfs-pod.yaml
```

```
deployment.apps/nginx-nfs created
```

```
student@lfs458-node-1a0a:~$ kubectl -n small describe deploy/nginx-nfs
```

```
Name:                nginx-nfs
Namespace:           small
<output_omitted>
```

26. Examine the pods to see if they are actually running.

```
student@lfs458-node-1a0a:~$ kubectl -n small get po
```

```
NAME                                READY   STATUS    RESTARTS   AGE
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@lfs458-node-1a0a:~$ kubectl -n small delete deploy nginx-nfs
```

```
deployment.apps "nginx-nfs" deleted
```

```
student@lfs458-node-1a0a:~$ kubectl -n small delete pvc/pvc-one
```

```
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@lfs458-node-1a0a:~$ kubectl -n small get pv
```

```
NAME      CAPACITY  ACCESSMODES  RECLAIMPOLICY  STATUS  CLAIM
STORAGECLASS  REASON    AGE
pvvol-1    1Gi      RWX          Delete         Failed  small/pvc-one  20m
```

29. Ensure the deployment, pvc and pv are all removed.

```
student@lfs458-node-1a0a:~$ kubectl delete pv/pvvol-1
```

```
persistentvolume "pvvol-1" deleted
```

30. Edit the persistent volume YAML file and change the persistentVolumeReclaimPolicy: to Recycle.

```
student@lfs458-node-1a0a:~$ vim PVol.yaml
```

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@lfs458-node-1a0a:~$ kubectl -n small create -f low-resource-range.yaml
```

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

32. View the settings for the namespace. Both quotas and resource limits should be seen.

```
student@lfs458-node-1a0a:~$ kubectl describe ns small
```



```
<output_omitted>
Resource Limits
Type      Resource  Min  Max  Default Request  Default Limit  ...
-----
Container cpu        -    -   500m             1              -
Container memory    -    -   100Mi           500Mi          -
```

33. Create the persistent volume again. View the resource. Note the Reclaim Policy is Recycle.

```
student@lfs458-node-1a0a:~$ kubectl -n small create -f PVol.yaml
persistentvolume/pvvol-1 created
```

```
student@lfs458-node-1a0a:~$ kubectl get pv
NAME      CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS  ...
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@lfs458-node-1a0a:~$ kubectl -n small create -f pvc.yaml
Error from server (Forbidden): error when creating "pvc.yaml":
persistentvolumeclaims "pvc-one" is forbidden: exceeded quota:
storagequota, requested: requests.storage=200Mi, used:
requests.storage=0, limited: requests.storage=100Mi
```

35. Edit the resourcequota to increase the requests.storage to 500mi.

```
student@lfs458-node-1a0a:~$ kubectl -n small edit resourcequota
```

YAML

```
1 ....
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@lfs458-node-1a0a:~$ kubectl -n small create -f pvc.yaml
persistentvolumeclaim/pvc-one created

student@lfs458-node-1a0a:~$ kubectl -n small create -f nfs-pod.yaml
deployment.apps/nginx-nfs created
```

37. View the namespace settings.

```
student@lfs458-node-1a0a:~$ kubectl describe ns small
<output_omitted>
```

38. Delete the deployment. View the status of the pv and pvc.

```
student@lfs458-node-1a0a:~$ kubectl -n small delete deploy nginx-nfs
deployment.apps "nginx-nfs" deleted
```

```
student@lfs458-node-1a0a:~$ kubectl -n small get pvc
```

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	AGE
pvc-one	Bound	pvvol-1	1Gi	RWX		7m

```
student@lfs458-node-1a0a:~$ kubectl -n small get pv
```

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM	...
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@lfs458-node-1a0a:~$ kubectl -n small delete pvc pvc-one
```

persistentvolumeclaim "pvc-one" deleted

```
student@lfs458-node-1a0a:~$ kubectl -n small get pv
```

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM	STORA...
pvvol-1	1Gi	RWX	Recycle	Available		...

40. Remove the pv and any other resources created during this lab.

```
student@lfs458-node-1a0a:~$ kubectl delete pv pvvol-1
```

persistentvolume "pvvol-1" deleted

Chapter 10

Ingress



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@lfs458-node-1a0a:~$ kubectl create deployment secondapp --image=nginx
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@lfs458-node-1a0a:~$ kubectl get deployments secondapp -o yaml |grep label -A2
labels:
  app: secondapp
  name: secondapp
--
labels:
  app: secondapp
spec:
```

3. Expose the new server as a NodePort.

```
student@lfs458-node-1a0a:~$ kubectl expose deployment secondapp \
  --type=NodePort --port=80
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@lfs458-node-1a0a:~$ vim ingress.rbac.yaml
```

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@lfs458-node-1a0a:~$ kubectl create -f ingress.rbac.yaml
clusterrole.rbac.authorization.k8s.io "traefik-ingress-controller" created
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@lfs458-node-1a0a:~$ find . -name traefik-ds.yaml
./LFS458/SOLUTIONS/s_10/traefik-ds.yaml

student@lfs458-node-1a0a:~$ cp ./LFS458/SOLUTIONS/s_10/traefik-ds.yaml .
```

7. Then create the ingress controller using **kubectl create**.

```
student@lfs458-node-1a0a:~$ kubectl create -f traefik-ds.yaml
serviceaccount "traefik-ingress-controller" created
daemonset.apps "traefik-ingress-controller" created
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@lfs458-node-1a0a:~$ vim ingress.rule.yaml
```

YAML

ingress.rule.yaml

```
1  apiVersion: networking.k8s.io/v1
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@lfs458-node-1a0a:~$ kubectl create -f ingress.rule.yaml
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@lfs458-node-1a0a:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc mq state UP group default qlen 1000
    link/ether 42:01:0a:80:00:03 brd ff:ff:ff:ff:ff:ff
    inet 10.128.0.7/32 brd 10.128.0.3 scope global ens4
        valid_lft forever preferred_lft forever
<output_omitted>

student@lfs458-node-1a0a:~$ curl -H "Host: www.example.com" http://k8smaster/
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
```

```
student@lfs458-node-1a0a:~$ curl -H "Host: www.example.com" http://35.193.3.179
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
<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@lfs458-node-1a0a:~$ kubectl create deployment thirdpage --image=nginx
deployment.apps/thirdpage created
```

12. Find the label for the new deployment. Look for the name:, which would be thirdpage in this example.

```
student@lfs458-node-1a0a:~$ kubectl get deployment thirdpage -o yaml |grep -A2 label

labels:
  app: thirdpage
  name: thirdpage
--
      labels:
        app: thirdpage
      spec:
```

13. Expose the new server as a NodePort.

```
student@lfs458-node-1a0a:~$ kubectl expose deployment \
  thirdpage --type=NodePort --port=80
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@lfs458-node-1a0a:~$ 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

<!DOCTYPE html>
<html>
<head>
<title>Third Page</title>
<style>

root@thirdpage-5cf8d67664-zcmfh:/# exit
```

15. Edit the ingress rules to point the thridpage service. Use the `serviceName` we found in an earlier step of thridpage.

```
student@lfs458-node-1a0a:~$ kubectl edit ingress ingress-test
```

YAML

```
1 <output_omitted>
2   - host: www.example.com
3     http:
4       paths:
5         - backend:
6             serviceName: secondapp
7             servicePort: 80
8         path: /
9   - host: thridpage.org
10    http:
11      paths:
12        - backend:
13            serviceName: thridpage
14            servicePort: 80
15        path: /
16  status:
17  <output_omitted>
```

16. Test the second hostname using **curl** locally as well as from a remote system.

```
student@lfs458-node-1a0a:~$ curl -H "Host: thridpage.org" http://k8smaster/
<!DOCTYPE html>
<html>
<head>
<title>Third Page</title>
<style>
<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 with a 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 and you can view traffic when you reference the pages, from inside or outside the node. Typo the domain names inside the **curl** command and you can also see 404 error traffic. Explore as time permits.

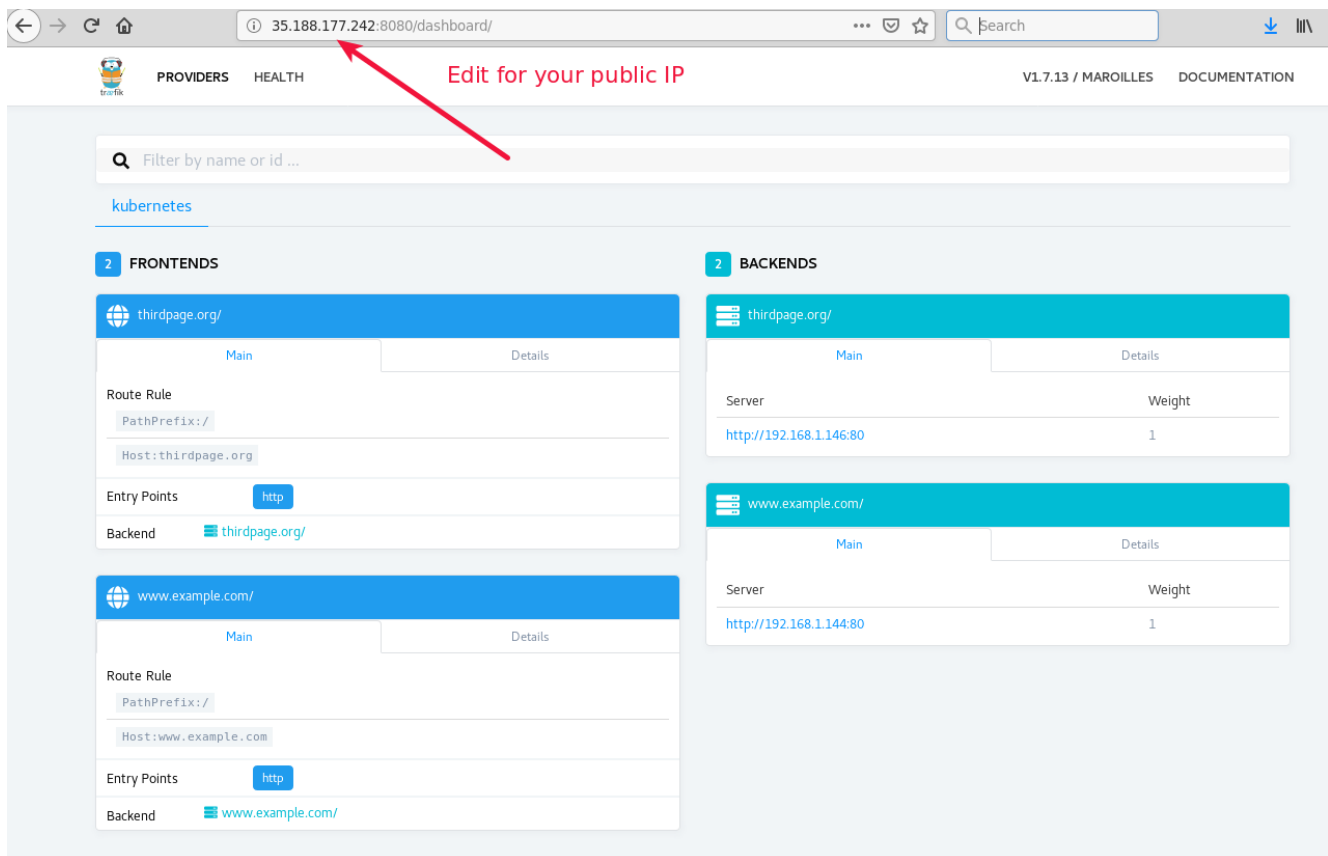


Figure 10.1: Accessing the API

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@lfs458-node-1a0a:~$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
lfs458-node-1a0a	Ready	master	44h	v1.16.1
lfs458-worker	Ready	<none>	43h	v1.16.1

2. View the current labels and taints for the nodes.

```
student@lfs458-node-1a0a:~$ kubectl describe nodes |grep -i label
```

Labels:	beta.kubernetes.io/arch=amd64
Labels:	beta.kubernetes.io/arch=amd64


```
student@lfs458-node-1a0a:~$ kubectl describe nodes |grep -i taint
```

Taints:	<none>
Taints:	<none>

3. Verify there are no deployments running, outside of the `kube-system` namespace. If there are, delete them. Then get a count of how many containers are running on both the master and secondary 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.

```
student@lfs458-node-1a0a:~$ kubectl get deployments --all-namespaces
```

NAMESPACE	NAME	READY	UP-TO-DATE	AVAILABLE	AGE
default	secondapp	1	1	1	37m
default	thirdpage	1	1	1	24m
kube-system	calico-kube-controllers	0	0	0	44h
kube-system	coredns	2/2	2	2	44h

```
student@lfs458-node-1a0a:~$ sudo docker ps |wc -l
```

```
21
```

```
student@lfs458-worker:~$ sudo docker ps |wc -l
```

```
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@lfs458-node-1a0a:~$ kubectl label nodes lfs458-node-1a0a status=vip
```

```
node/lfs458-node-1a0a labeled
```

```
student@lfs458-node-1a0a:~$ kubectl label nodes lfs458-worker status=other
```

```
node/lfs458-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@lfs458-node-1a0a:~$ kubectl get nodes --show-labels
```

NAME	STATUS	ROLES	AGE	VERSION	LABELS
lfs458-node-1a0a	Ready	master	44h	v1.16.1	beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/hostname=lfs458-node-1a0a,node-role.kubernetes.io/master=,status=vip
lfs458-worker	Ready	<none>	44h	v1.16.1	beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/hostname=lfs458-worker,status=other

6. Create `vip.yaml` to spawn four busybox containers which sleep the whole time. Include the `nodeSelector` entry.

```
student@lfs458-node-1a0a:~$ vim vip.yaml
```

YA
ML

vip.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@lfs458-node-1a0a:~$ kubectl create -f vip.yaml
pod/vip created

```

```

student@lfs458-node-1a0a:~$ sudo docker ps |wc -l
26

```

```

student@lfs458-worker:~$ sudo docker ps |wc -l
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@lfs458-node-1a0a:~$ kubectl delete pod vip
pod "vip" deleted

```

```

student@lfs458-node-1a0a:~$ vim vip.yaml
....
#   nodeSelector:
#       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@lfs458-node-1a0a:~$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
secondapp-85765cd95c-2q9sx         1/1     Running   0           43m
thirdpage-7c9b56bfd-2q5pr          1/1     Running   0           30m

```

```

student@lfs458-node-1a0a:~$ kubectl create -f vip.yaml
pod/vip created

```

10. Determine where the new containers have been deployed. They should be more evenly spread this time.

```

student@lfs458-node-1a0a:~$ sudo docker ps |wc -l
21

```

```

student@lfs458-worker:~$ sudo docker ps |wc -l
16

```

11. Create another file for other users. Change the names from vip to others, and uncomment the nodeSelector lines.

```

student@lfs458-node-1a0a:~$ cp vip.yaml other.yaml

student@lfs458-node-1a0a:~$ sed -i s/vip/other/g other.yaml

student@lfs458-node-1a0a:~$ vim other.yaml

```

YAML

other.yaml

```

1  ....
2  nodeSelector:
3  status: other

```

12. Create the other containers. Determine where they deploy.

```

student@lfs458-node-1a0a:~$ kubectl create -f other.yaml
pod/other created

```

```

student@lfs458-node-1a0a:~$ sudo docker ps |wc -l
21

```

```

student@lfs458-worker:~$ sudo docker ps |wc -l
21

```

13. Shut down both pods and verify they terminated. Only our previous pods should be found.

```

student@lfs458-node-1a0a:~$ kubectl delete pods vip other
pod "vip" deleted
pod "other" deleted

```

```

student@lfs458-node-1a0a:~$ kubectl get pods

```

NAME	READY	STATUS	RESTARTS	AGE
secondapp-85765cd95c-2q9sx	1/1	Running	0	51m
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@lfs458-node-1a0a:~$ kubectl delete deployment secondapp \
thirdpage
deployment.apps "secondapp" deleted
deployment.apps "thirdpage" deleted

```

2. Create a deployment which will deploy eight **nginx** containers. Begin by creating a YAML file.

```

student@lfs458-node-1a0a:~$ vim taint.yaml

```

YAML

taint.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:1.11.1
18        ports:
19      - containerPort: 80

```

3. Apply the file to create the deployment.

```

student@lfs458-node-1a0a:~$ kubectl apply -f taint.yaml
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@lfs458-node-1a0a:~$ sudo docker ps |grep nginx
00c1be5df1e7      nginx@sha256:e3456c851a152494c3e.....
<output_omitted>

```

```

student@lfs458-node-1a0a:~$ sudo docker ps |wc -l
27

```

```

student@lfs458-worker:~$ sudo docker ps |wc -l
17

```

5. Delete the deployment. Verify the containers are gone.

```

student@lfs458-node-1a0a:~$ kubectl delete deployment taint-deployment
deployment.apps "taint-deployment" deleted

```

```

student@lfs458-node-1a0a:~$ sudo docker ps |wc -l
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@lfs458-node-1a0a:~$ kubectl taint nodes lfs458-worker \
    bubba=value:PreferNoSchedule
node/lfs458-worker tainted

```

```

student@lfs458-node-1a0a:~$ kubectl describe node |grep Taint

```

```
Taints:                bubba=value:PreferNoSchedule
Taints:                <none>
```

```
student@lfs458-node-1a0a:~$ kubectl apply -f taint.yaml
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@lfs458-node-1a0a:~$ sudo docker ps |wc -l
21
```

```
student@lfs458-worker:~$ sudo docker ps |wc -l
23
```

```
student@lfs458-node-1a0a:~$ kubectl delete deployment taint-deployment
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@lfs458-node-1a0a:~$ kubectl taint nodes lfs458-worker bubba-
node/lfs458-worker untainted
```

```
student@lfs458-node-1a0a:~$ kubectl describe node |grep Taint
Taints:                <none>
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@lfs458-node-1a0a:~$ kubectl taint nodes lfs458-worker \
    bubba=value:NoSchedule
node/lfs458-worker tainted
```

```
student@lfs458-node-1a0a:~$ kubectl apply -f taint.yaml
deployment.apps/taint-deployment created
```

```
student@lfs458-node-1a0a:~$ sudo docker ps |wc -l
21
```

```
student@lfs458-worker:~$ sudo docker ps |wc -l
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@lfs458-node-1a0a:~$ kubectl delete deployment taint-deployment
deployment.apps "taint-deployment" deleted
```

```
student@lfs458-node-1a0a:~$ kubectl taint nodes lfs458-worker bubba-
node/lfs458-worker untainted
```

```
student@lfs458-node-1a0a:~$ kubectl apply -f taint.yaml
deployment.apps/taint-deployment created
```

```
student@lfs458-node-1a0a:~$ sudo docker ps |wc -l
27
```

```
student@lfs458-worker:~$ sudo docker ps |wc -l
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@lfs458-node-1a0a:~$ kubectl taint nodes lfs458-worker \
    bubba=value:NoExecute
node "lfs458-worker" tainted
```

```
student@lfs458-node-1a0a:~$ sudo docker ps |wc -l
37
```

```
student@lfs458-worker:~$ sudo docker ps |wc -l
5
```

12. Remove the taint. Wait a minute. Note that all of the containers did not return to their previous placement.

```
student@lfs458-node-1a0a:~$ kubectl taint nodes lfs458-worker bubba-
node/lfs458-worker untainted
```

```
student@lfs458-node-1a0a:~$ sudo docker ps |wc -l
32
```

```
student@lfs458-worker:~$ sudo docker ps |wc -l
6
```

13. In addition to the ability to taint a node you can also set the status to drain. First view the status, then destroy the existing deployment. Note that the status reports Ready, even though it will not allow containers to be executed. Also note that the output mentioned that DaemonSet-managed pods are not affected by default, as we saw in an earlier lab. This time lets take a closer look at what happens to existing pods and nodes.

Existing containers are not moved, but no new containers are created. You may receive an error
error unable to drain node "your node", aborting command....:

```
student@lfs458-node-1a0a:~$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
lfs458-node-1a0a	Ready	master	44h	v1.16.1
lfs458-worker	Ready	<none>	44h	v1.16.1

```
student@lfs458-node-1a0a:~$ kubectl drain lfs458-worker
```

```
node/lfs458-worker cordoned
error: unable to drain node "lfs458-worker", aborting command...
```

```
There are pending nodes to be drained:
```

```
lfs458-worker
```

```
error: cannot delete DaemonSet-managed Pods (use --ignore-daemonsets to ignore):kube-system/calico-node-dr279, kube-system/traefik-ingress-controller-mcn47
```

14. Verify the state change of the node. It should indicate no new Pods will be scheduled.

```
student@lfs458-node-1a0a:~$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
lfs458-node-1a0a	Ready	master	44h	v1.16.1
lfs458-worker	Ready,SchedulingDisabled	<none>	44h	v1.16.1

15. Delete the deployment to destroy the current Pods.

```
student@lfs458-node-1a0a:~$ kubectl delete deployment taint-deployment
deployment.apps "taint-deployment" deleted
```

16. Create the deployment again and determine where the containers have been deployed.

```
student@lfs458-node-1a0a:~$ kubectl apply -f taint.yaml
deployment.apps/taint-deployment created
```

```
student@lfs458-node-1a0a:~$ sudo docker ps |wc -l
37
```

17. Return the status to Ready, then destroy and create the deployment again. The containers should be spread across the nodes. Begin by removing the cordon on the node.

```
student@lfs458-node-1a0a:~$ kubectl uncordon lfs458-worker
node/lfs458-worker uncordoned
```

```
student@lfs458-node-1a0a:~$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
lfs458-node-1a0a	Ready	master	44h	v1.16.1
lfs458-worker	Ready	<none>	44h	v1.16.1

18. Delete and re-create the deployment.

```
student@lfs458-node-1a0a:~$ kubectl delete deployment taint-deployment
deployment.apps "taint-deployment" deleted
```

```
student@lfs458-node-1a0a:~$ kubectl apply -f taint.yaml
deployment.apps/taint-deployment created
```

19. View the **docker ps** output again. Both nodes should have almost the same number of containers deployed. The master will have a few more, due to its role.

20. Remove the deployment a final time to free up resources.

```
student@lfs458-node-1a0a:~$ kubectl delete deployment taint-deployment
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@lfs458-node-1a0a:~$ journalctl -u kubelet |less
<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@lfs458-node-1a0a:~$ sudo find / -name "*apiserver*log"
/var/log/containers/kube-apiserver-u16-12-1-dcb8_kube-system_kube-apiserver-
eddae7079382cd382cd55f8f46b192565dd16b6858206039d49b1ad4693c2a10.log
/var/log/containers/kube-apiserver-u16-12-1-dcb8_kube-system_kube-apiserver-
d00a48877af4ed4c7f8eedf2c7805c77cfabb31fcb453f7d89ffa52fc6ea5f36.log
```

```
student@lfs458-node-1a0a:~$ sudo diff /var/log/containers/kube-apiserver-u16-12-1-dcb8_kube-system_kube-apiserver-eddae7079382cd382cd55f8f46b192565dd16b6858206039d49blad4693c2a10.log /var/log/containers/kube-apiserver-u16-12-1-dcb8_kube-system_kube-apiserver-d00a48877af4ed4c7f8eedf2c7805c77cfabb31fc453f7d89ffa52fc6ea5f36.log
```

<output_omitted>

3. Take a look at the log file.

```
student@lfs458-node-1a0a:~$ sudo less /var/log/containers/kube-apiserver-u16-12-1-dcb8_kube-system_kube-apiserver-d00a48877af4ed4c7f8eedf2c7805c77cfabb31fc453f7d89ffa52fc6ea5f36.log
```

<output_omitted>

4. Search for and review other log files for kube-dns, kube-flannel, and kube-proxy.
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@lfs458-node-1a0a:~$ kubectl get po --all-namespaces
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
kube-system	calico-kube-controllers-7b9dcdcc5-qg6zd	1/1	Running	0	13m
kube-system	calico-node-dr279	1/1	Running	0	6d1h
....					
kube-system	etcd-lfs458-node-1a0a	1/1	Running	2	44h
kube-system	kube-apiserver-lfs458-node-1a0a	1/1	Running	2	44h
kube-system	kube-controller-manager-lfs458-node-1a0a	1/1	Running	2	44h
kube-system	kube-scheduler-lfs458-node-1a0a	1/1	Running	2	44h
....					

- 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@lfs458-node-1a0a:~$ kubectl -n kube-system logs <Tab><Tab>
```

```
calico-kube-controllers-7b9dcdcc5-qg6zd
calico-node-dr279
calico-node-xtvfd
coredns-5644d7b6d9-k7kts
coredns-5644d7b6d9-rnr2v
etcd-lfs458-node-1a0a
kube-apiserver-lfs458-node-1a0a
kube-controller-manager-lfs458-node-1a0a
kube-proxy-qhc4f
kube-proxy-s56hl
kube-scheduler-f-lfs458-node-1a0a
traefik-ingress-controller-hw5tv
traefik-ingress-controller-mcn47
```

```
student@lfs458-node-1a0a:~$ kubectl -n kube-system logs \
    kube-apiserver-lfs458-node-1a0a
```

```
Flag --insecure-port has been deprecated, This flag will be removed in a future version.
I1119 02:31:14.933023      1 server.go:623] external host was not specified, using 10.128.0.3
I1119 02:31:14.933356      1 server.go:149] Version: v1.16.1
I1119 02:31:15.595131      1 plugins.go:158] Loaded 11 mutating admission controller(s)
successfully in the following order: NamespaceLifecycle,LimitRanger,ServiceAccount,
NodeRestriction,TaintNodesByCondition,Priority,DefaultTolerationSeconds,DefaultStorageClass,
StorageObjectInUseProtection,MutatingAdmissionWebhook,RuntimeClass.
I1119 02:31:15.595357      1 plugins.go:161] Loaded 7 validating admission controller(s)
successfully in the following order: LimitRanger,ServiceAccount,Priority,
PersistentVolumeClaimResize,ValidatingAdmissionWebhook,RuntimeClass,
ResourceQuota.
<output_omitted>
```

- 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

- Begin by cloning the software. The **git** command should be installed already. Install it if not found.

```
student@lfs458-node-1a0a:~$ git clone \
    https://github.com/kubernetes-incubator/metrics-server.git
```

```
<output_omitted>
```

2. Create the necessary objects.

```
student@lfs458-node-1a0a:~$ kubectl create -f metrics-server/deploy/1.8+/
clusterrole.rbac.authorization.k8s.io/system:aggregated-metrics-reader created
clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-delegator created
rolebinding.rbac.authorization.k8s.io/metrics-server-auth-reader created
apiservice.apiregistration.k8s.io/v1beta1.metrics.k8s.io created
serviceaccount/metrics-server created
deployment.apps/metrics-server created
service/metrics-server created
clusterrole.rbac.authorization.k8s.io/system:metrics-server created
clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server created
```

3. View the current objects, which are created in the kube-system namespace. All should show a Running status.

```
student@lfs458-node-1a0a:~$ kubectl -n kube-system get pods
<output_omitted>
kube-proxy-ld2hb                1/1      Running    0          2d21h
kube-scheduler-u16-1-13-1-2f8c  1/1      Running    0          2d21h
metrics-server-fc6d4999b-b9rjj  1/1      Running    0          42s
```

4. 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.

```
student@lfs458-node-1a0a:~$ kubectl -n kube-system edit deployment metrics-server
```

YAML

```
1 ....
2     spec:
3       containers:
4         - args:
5           - --cert-dir=/tmp
6           - --secure-port=4443
7           - --kubelet-insecure-tls          #<-- Add this and following line
8           - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
9           image: k8s.gcr.io/metrics-server-amd64:v0.3.6
10          imagePullPolicy: Always
11          name: metrics-server
12 ....
```

5. Test that the metrics server pod is running and does now show errors. You should see about five lines showing the container is listening.

```
student@lfs458-node-1a0a:~$ kubectl -n kube-system logs metrics-server<TAB>
I0110 20:21:27.333663      1 serving.go:273] Generated self-signed cert
(apiserver.local.config/certificates/apiserver.crt, apiserver.local.config
/certificates/apiserver.key)
[restful] 2019/01/10 20:21:28 log.go:33: [restful/swagger] listing is
available at https://:443/swaggerapi
[restful] 2019/01/10 20:21:28 log.go:33: [restful/swagger] https://:443/
swaggerui/ is mapped to folder /swagger-ui/
I0110 20:21:28.539435      1 serve.go:96] Serving securely on [::]:443
```

6. 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@lfs458-node-1a0a:~$ kubectl top pod --all-namespaces
```

```

NAMESPACE      NAME                                         CPU(cores)   MEMORY(bytes)
kube-system     calico-kube-controllers-7b9dcdcc5-qg6zd    2m           6Mi
kube-system     calico-node-dr279                          23m          22Mi
kube-system     calico-node-xtvfd                          21m          22Mi
kube-system     coredns-5644d7b6d9-k7kts                  2m           6Mi
kube-system     coredns-5644d7b6d9-rnr2v                  3m           6Mi
<output_omitted>

```

```
student@lfs458-node-1a0a:~$ kubectl top nodes
```

```

NAME                CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
lfs458-node-1a0a    228m        11%    2357Mi          31%
lfs458-worker       76m         3%     1385Mi          18%

```

7. Using keys we generated in an earlier lab we can also interrogate the API server. Your server IP address will be different.

```

student@lfs458-node-1a0a:~$ 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>

```

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@lfs458-node-1a0a:~$ 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@lfs458-node-1a0a:~$ kubectl get svc --all-namespaces
```

```
<output_omitted>
```

default	thirdpage	NodePort	10.101.190.109	<none>	80:30588/TCP
kube-system	kube-dns	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@lfs458-node-1a0a:~$ kubectl -n kubernetes-dashboard edit svc kubernetes-dashboard
```

YAML

```
1 ....
2 selector:
3   k8s-app: kubernetes-dashboard
4 sessionAffinity: None
5 type: NodePort                                #<-- Edit this line
6 status:
7 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@lfs458-node-1a0a:~$ 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@lfs458-node-1a0a:~$ kubectl create clusterrolebinding dashaccess \
--clusterrole=cluster-admin \
--serviceaccount=kubernetes-dashboard:kubernetes-dashboard
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.

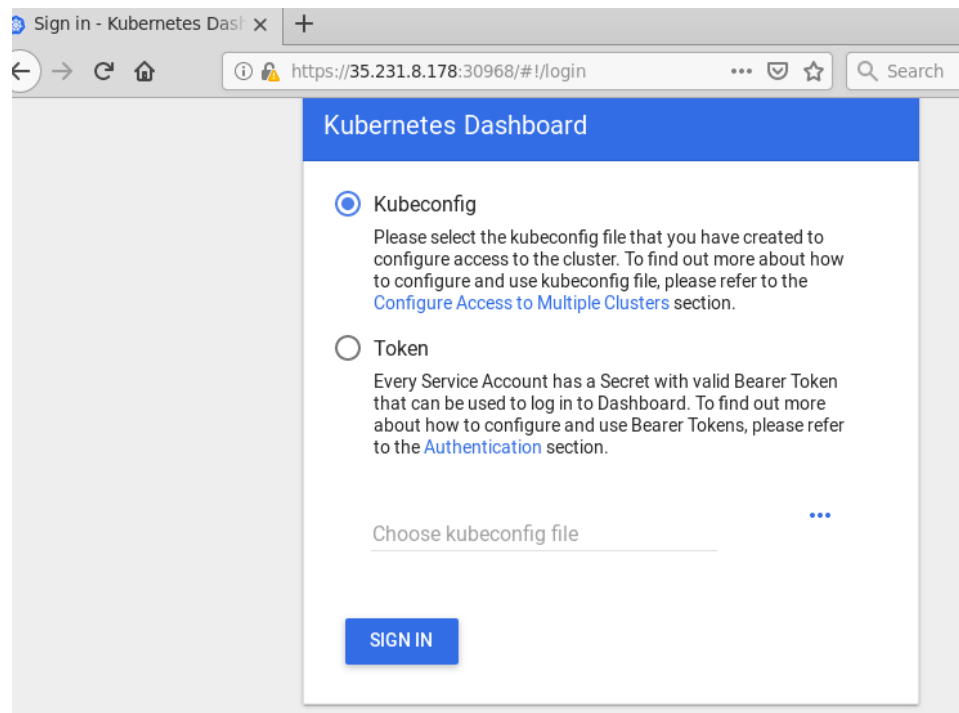


Figure 12.1: External Access via Browser

- 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@lfs458-node-1a0a:~$ kubectl -n kube-system describe secrets kubernetes-dashboard-token-<Tab>
```

```
....
```

```
Data
```

```
====
```

```
ca.crt:      1025 bytes
```

```
namespace:  11 bytes
```

```
token:       eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJ1cm5ldGVzL3N1cnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWN1YWNjb3VudC9uYWw1c3BhY2UiOiJrdWJ1c3R1bSIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJrdWJ1cm5ldGVzLWRhc2hib2FyZC10b2t1bi1wbW04NCIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW50Lm5hbWUiOiJrdWJ1cm5ldGVzLWRhc2hib2FyZCIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6IjE5MDY4ZDIzLTE1MTctMTF0S1hZmMyLTQyMDEwYThlMDAwMyIsInN1YiI6InN5c3R1bTpxZXJ2aWN1YWNjb3VudDprdwJ1c3R1bTprdwJ1cm5ldGVzLWRhc2hib2FyZCJ9.aYTUMWr290pjt5i32rb8qXpq4onn3hLhvz6yLSYexgRd6NysygvUyqnkRsFE1trg9i1ftNXKJdzkY5kQzN3AcpUTvyj_BvJgzNh3JM9p7QMjI8LHTz4TrRZrvwJVWitrEn4VnTQuFVcADFD_rKB9FyI_gvT_QiW5fQm24ygTlgfOYd44263oakG8sL64q7UfQNW2wt5S0orMutybOmX4CXNUYM8G44ejEtv9GW50sVjEmLIGaoEMX7fctwUN_XCyPdzcCg2W0xRHahBjmbCuLz2SSWL52q4nXQmhTq_L8VDDpt6LjEqXW6LtDJZGjVCs2MnBLerQz-ZAgsVaubbQ
```

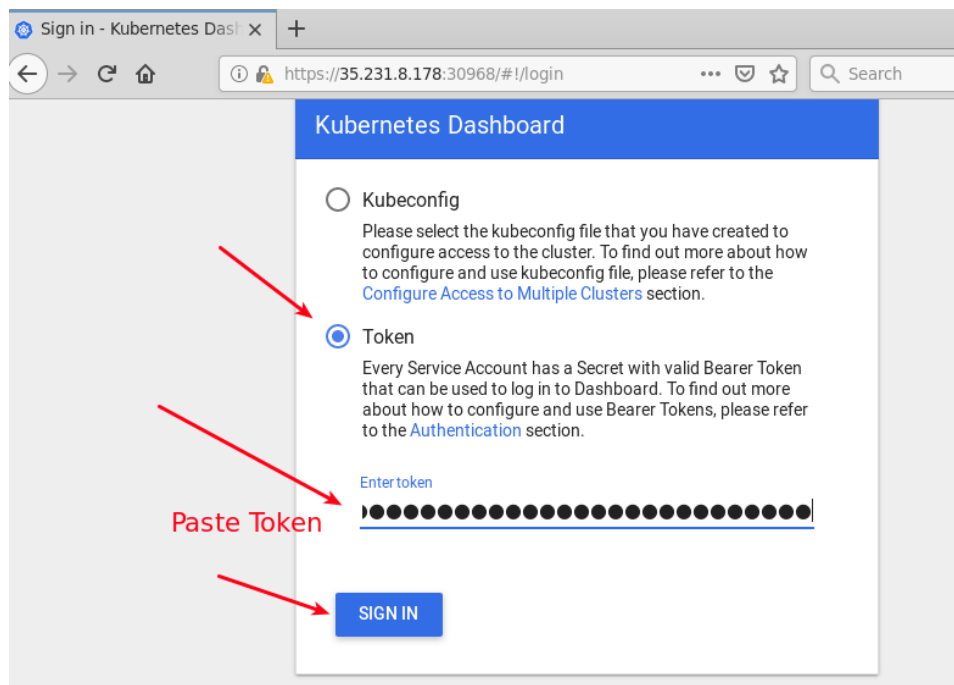


Figure 12.2: External Access via Browser

8. Navigate around the various sections and use the menu to the left as time allows.

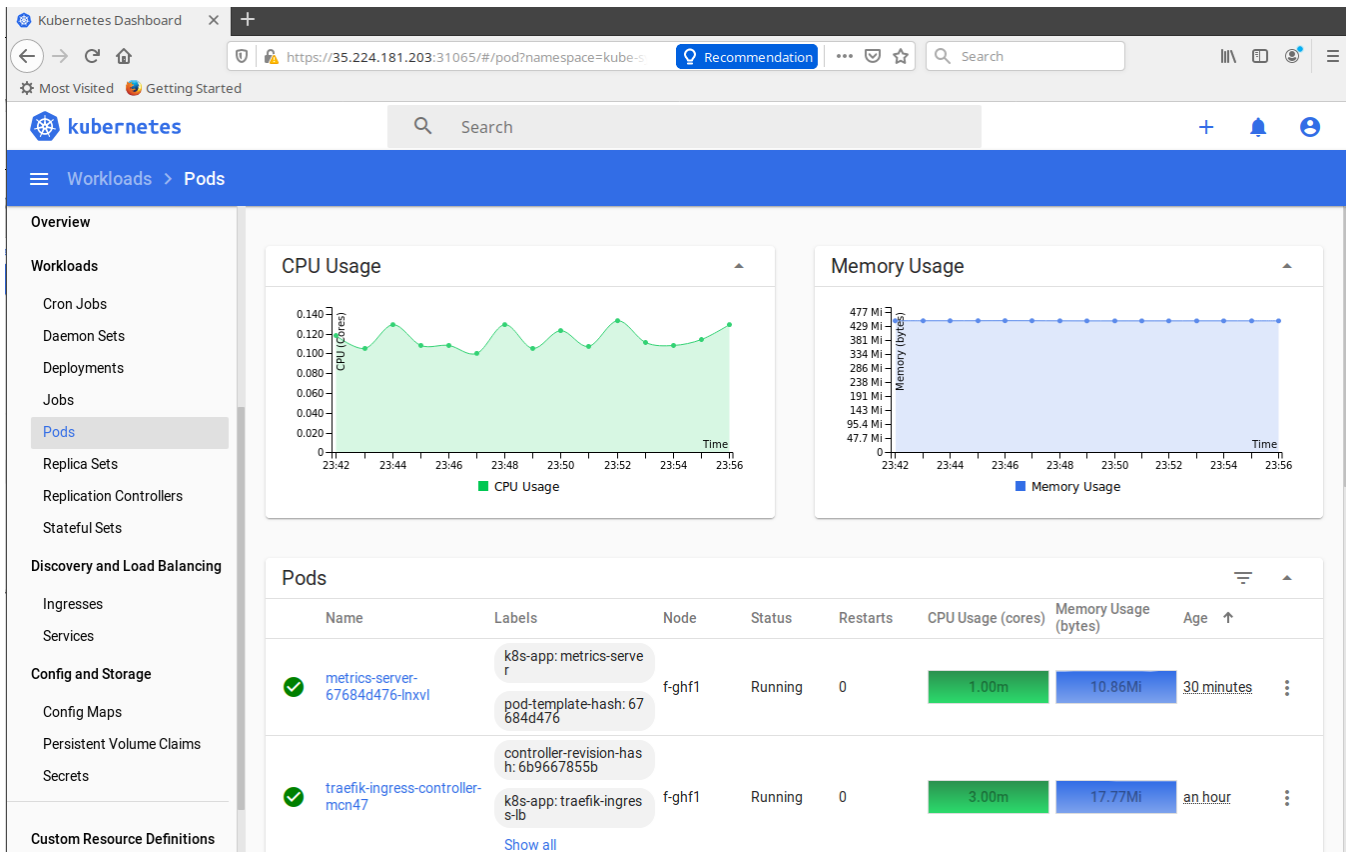


Figure 12.3: External Access via Browser

Chapter 13

Custom Resource Definition



13.1 Labs

Exercise 13.1: Create a Custom Resource Definition

Overview

ThirdPartyResource is no longer included with the API in v1.8 and its use will return a validation error. If you have upgraded from a version prior to Kubernetes v1.7, you will need to convert them to CustomResourceDefinitions (CRD). A new resource often requires a controller to manage the resource. Creation of the controller is beyond the scope of this course, basically it is a watch-loop comparing a spec file to the current state 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>.

We will 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. We will create a new YAML file.

```
student@lfs458-node-1a0a:~$ vim crd.yaml
```

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

```

2. Add the new resource to the cluster.

```

student@lfs458-node-1a0a:~$ kubectl create -f crd.yaml
customresourcedefinition.apiextensions.k8s.io/crontabs.training.lfs458.com
created

```

3. View and describe the resource. You'll note the **describe** output is unlike other objects we have seen so far.

```

student@lfs458-node-1a0a:~$ kubectl get crd

NAME                                CREATED AT
crontabs.training.lfs458.com        2018-08-03T05:25:20Z
<output_omitted>

student@lfs458-node-1a0a:~$ kubectl describe crd crontab<Tab>

Name:                crontabs.training.lfs458.com
Namespace:
Labels:              <none>
Annotations:         <none>
API Version:         apiextensions.k8s.io/v1beta1
Kind:                CustomResourceDefinition
<output_omitted>

```

4. 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@lfs458-node-1a0a:~$ vim new-crontab.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

```

5. Create the new object and view the resource using short and long name.

```

student@lfs458-node-1a0a:~$ kubectl create -f new-crontab.yaml
crontab.training.lfs458.com/new-cron-object created

student@lfs458-node-1a0a:~$ kubectl get CronTab

NAME            AGE
new-cron-object  22s

```

```
student@lfs458-node-1a0a:~$ kubectl get ct

NAME                AGE
new-cron-object     29s

student@lfs458-node-1a0a:~$ kubectl describe ct

Name:                new-cron-object
Namespace:
Labels:              <none>

<output_omitted>

Spec:
  Cron Spec:  */5 * * * *
  Image:      some-cron-image
  Events:     <none>
```

6. To clean up the resources we will delete the CRD. This should delete all of the endpoints and objects using it as well.

```
student@lfs458-node-1a0a:~$ kubectl delete -f crd.yaml

customresourcedefinition.apiextensions.k8s.io
"crontabs.training.lfs458.com" deleted

student@lfs458-node-1a0a:~$ kubectl get ct

Error from server (NotFound): Unable to list "crontabs": 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@lfs458-node-1a0a:~$ wget https://get.helm.sh/helm-v3.0.0-linux-amd64.tar.gz
```

```
<output_omitted>
```

```
helm-v3.0.0-linux-a 100%[=====] 11.52M --.-KB/s in 0.1s
```

```
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@lfs458-node-1a0a:~$ tar -xvf helm-v3.0.0-linux-amd64.tar.gz
```

```
linux-amd64/
```

```
linux-amd64/helm
```

```
linux-amd64/README.md
```

```
linux-amd64/LICENSE
```

3. Copy the **helm** binary to the `/usr/local/bin/` directory, so it is usable via the shell search path.

```
student@lfs458-node-1a0a:~$ sudo cp linux-amd64/helm /usr/local/bin/helm3
```

4. A Chart is a collection of files to deploy an application. There is a collection 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.

```
student@lfs458-node-1a0a:~$ helm3 search hub database
```

URL	CHART VERSION	APP VERSION	DESCRIPTION
https://hub.helm.sh/charts/bitnami/postgresql	7.6.0	11.6.0	Chart for Post...
https://hub.helm.sh/charts/bitnami/redis	10.0.1	5.0.7	Open source, a...
https://hub.helm.sh/charts/bitnami/etcd	4.4.4	3.4.3	etcd is a dist...
https://hub.helm.sh/charts/bitnami/mongodb	7.4.6	4.0.13	NoSQL document...
https://hub.helm.sh/charts/bitnami/cassandra	4.1.8	3.11.5	Apache Cassand...
https://hub.helm.sh/charts/bitnami/dokuwiki	6.0.2	0.20180422.201901061035	Dok...
https://hub.helm.sh/charts/bitnami/influxdb	0.2.3	1.7.9	InfluxDB is an...
https://hub.helm.sh/charts/bitnami/mariadb	7.0.1	10.3.20	Fast, reliable...

<output_omitted>

5. We will begin by adding a common repository and call it stable.

```
student@lfs458-node-1a0a:~$ helm3 repo add stable https://kubernetes-charts.storage.googleapis.com
"stable" has been added to your repositories
```

```
student@lfs458-node-1a0a:~$ helm3 repo update
```

```
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "stable" chart repository
Update Complete. Happy Helming!
```

6. We will install the **mariadb**. The **-debug** option will create a lot of output. Generate an interesting name for the deployment, like **illmannered-salamander**. 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 use to create an available PV.

```
student@lfs458-node-1a0a:~$ helm3 --debug install firstdb stable/mariadb \
--set master.persistence.enabled=false \
--set slave.persistence.enabled=false
```

```
install.go:148: [debug] Original chart version: ""
install.go:165: [debug] CHART PATH: /home/student/.cache/helm/repository/mariadb-7.0.1.tgz

client.go:87: [debug] creating 8 resource(s)
NAME: firstdb
LAST DEPLOYED: Mon Nov 25 08:03:30 2019
<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@lfs458-node-1a0a:~$ kubectl get secret --namespace default firstdb-mariadb \
-o jsonpath="{.data.mariadb-root-password}" | base64 --decode
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@lfs458-node-1a0a:~$ kubectl run firstdb-mariadb-client \
--rm --tty -i --restart='Never' \
--image docker.io/bitnami/mariadb:10.3.20-debian-9-r0 \
--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

Welcome to the MariaDB monitor.  Commands end with ; or \ g.
Your MariaDB connection id is 153
Server version: 10.1.38-MariaDB Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\ h' for help. Type '\ c' to clear the current input statement.

MariaDB [(none)]> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| my_database |
| mysql |
| performance_schema |
| test |
+-----+
5 rows in set (0.00 sec)

MariaDB [(none)]>
MariaDB [(none)]> quit
Bye
I have no name!@firstdb-mariadb-client:/$ exit
```

10. View the Chart history on the system. The use of the **-a** option will show all Charts including deleted and failed attempts. The output below shows the current running Chart as well as a previously deleted **hadoop** Chart. So you can see previous installations.

```
student@lfs458-node-1a0a:~$ helm3 list

NAME          NAMESPACE      REVISION      UPDATED
STATUS        CHART           APP VERSION
firstdb       default         1             2019-11-25 08:03:30.693410615
+0000 UTCdeployed      mariadb-7.0.1    10.3.20
```

11. Delete the **mariadb** Chart. No releases of mariadb should be found.

```
student@lfs458-node-1a0a:~$ helm3 uninstall firstdb
release "firstdb" uninstalled
```

```
student@lfs458-node-1a0a:~$ helm3 list
```

```
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@lfs458-node-1a0a:~$ find ~ -name *mariadb*
/home/student/.cache/helm/repository/mariadb
/home/student/.cache/helm/repository/mariadb-7.0.1.tgz
```

13. Move to the archive directory and extract the tarball. Take a look at the files within.

```
student@lfs458-node-1a0a:~$ cd ~/.cache/helm/repository ; tar -xvf mariadb-*
mariadb/Chart.yaml
mariadb/values.yaml
mariadb/templates/NOTES.txt
mariadb/templates/_helpers.tpl
mariadb/templates/initialization-configmap.yaml
<output_omitted>
```

14. Copy and rename the `values.yaml` file back to the home directory and return there as well.

```
student@lfs458-node-1a0a:~$ cp mariadb/values.yaml ~/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@lfs458-node-1a0a:~$ less custom.yaml
<output_omitted>
```

16. Edit the file. We will change or add two items, the `rootUser` password and storage persistence parameters.

```
student@lfs458-node-1a0a:~$ vim custom.yaml
....
rootUser:
  ## MariaDB admin password
  ## ref: https://github.com/bitnami/bitnami-docker-mariadb#setting-the-root-password-on-first-run
  ##
  password: LFTr@1n                                #<-- Add a password, such as LFTr@1n
  ##
....
persistence:
  ## If true, use a Persistent Volume Claim, If false, use emptyDir
  ##
  enabled: false                                #<--- Change this to false
  # Enable persistence using an existing PVC
  # existingClaim:
....
```

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. Your instance name will be different. Near the end of the output will be commands on using the database.

```
student@lfs458-node-1a0a:~$ helm3 install -f values.yaml stable/mariadb
NAME: seconddb
LAST DEPLOYED: Mon Nov 25 08:54:08 2019
NAMESPACE: default
STATUS: deployed
```

<output_omitted>

To connect to your database:

1. Run a pod that you can use as a client:

```
kubectl run seconddb-mariadb-client --rm --tty -i --restart='Never' --image
docker.io/bitnami/mariadb:10.3.20-debian-9-r0 --namespace default --command -- bash
```

2. To connect to master service (read/write):

<output_omitted>

18. Copy and paste the command to run a client pod from the previous command output. If you are typing, don't include the backslash. The command could be on a single line. Once in the container log into the database with the password we set and check the default databases.

```
student@lfs458-node-1a0a:~$ kubectl run seconddb-mariadb-client --rm --tty \
-i --restart='Never' --image docker.io/bitnami/mariadb:10.3.20-debian-9-r0 \
--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: LFTTr@1n #<-- The password we set in the config file
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 25
Server version: 10.3.17-MariaDB-log Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [my_database]> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| my_database |
| mysql |
| performance_schema |
| test |
+-----+
5 rows in set (0.001 sec)

MariaDB [my_database]> quit
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@lfs458-node-1a0a:~$ systemctl status kubelet.service
kubelet.service - kubelet: The Kubernetes Node Agent
   Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor preset: en
   Drop-In: /etc/systemd/system/kubelet.service.d
            |_10-kubeadm.conf
<output_omitted>
```

2. If we look at the status output, and follow the cgroup information, which is a long line we where configuration settings are drawn from, we see where the configuration file can be found.

```
CGroup: /system.slice/kubelet.service
|--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@lfs458-node-1a0a:~$ sudo less /var/lib/kubelet/config.yaml
```

YAML
config.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@lfs458-node-1a0a:~$ sudo ls /etc/kubernetes/manifests/
```

```
etcd.yaml           kube-controller-manager.yaml
kube-apiserver.yaml kube-scheduler.yaml
```

```
student@lfs458-node-1a0a:~$ sudo less \
    /etc/kubernetes/manifests/kube-controller-manager.yaml
```

```
<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@lfs458-node-1a0a:~$ kubectl -n kube-system get secrets
```

```
NAME                                TYPE
DATA      AGE
attachdetach-controller-token-xqr8n  kubernetes.io/service-account-token
3        5d
bootstrap-signer-token-xbp6s         kubernetes.io/service-account-token
3        5d
bootstrap-token-i3r13t               bootstrap.kubernetes.io/token
7        5d
<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@lfs458-node-1a0a:~$ kubectl -n kube-system get secrets \
    certificate<Tab> -o yaml
```

YAML

```
1 apiVersion: v1
2 data:
3   ca.crt: LS0tLS1CRUdJTi....
4   namespace: a3ViZS1zeXNOZW0=
5   token: ZX1KaGJHY2lPaUpTVXpJM....
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 **kubect**l **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@lfs458-node-1a0a:~$ kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: REDACTED
<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@lfs458-node-1a0a:~$ kubectl config set-credentials -h
Sets a user entry in kubeconfig
<output_omitted>

```

9. Make a copy of your access configuration file. Later steps will update this file and we can view the differences.

```

student@lfs458-node-1a0a:~$ cp ~/.kube/config ~/.cluster-api-config

```

10. Explore working with cluster and security configurations both using **kubect**l and **kubeadm**. Among other values, find the name of your cluster. You will need to become root to work with **kubeadm**.

```

student@lfs458-node-1a0a:~$ kubectl config <Tab><Tab>
current-context  get-contexts    set-context     view
delete-cluster  rename-context  set-credentials
delete-context   set             unset
get-clusters     set-cluster     use-context

```

```

student@lfs458-node-1a0a:~$ sudo kubeadm token -h
<output_omitted>

```

```

student@lfs458-node-1a0a:~$ sudo kubeadm config -h
<output_omitted>

```

11. Review the cluster default configuration settings. There may be some interesting tidbits to the security and infrastructure of the cluster.

```

student@lfs458-node-1a0a:~$ sudo kubeadm config print init-defaults
apiVersion: kubeadm.k8s.io/v1beta2
bootstrapTokens:
- groups:
  - system:bootstrappers:kubeadm:default-node-token
  token: abcdef.0123456789abcdef
  ttl: 24h0m0s
  usages:
<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@lfs458-node-1a0a:~$ kubectl create ns development
namespace/development created
```

```
student@lfs458-node-1a0a:~$ kubectl create ns production
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@lfs458-node-1a0a:~$ kubectl config get-contexts
```

CURRENT	NAME	CLUSTER	AUTHINFO	NAMESPACE
*	kubernetes-admin@kubernetes	kubernetes	kubernetes	kubernetes-admin

3. Create a new user DevDan and assign a password of lfs458.

```
student@lfs458-node-1a0a:~$ sudo useradd -s /bin/bash DevDan

student@lfs458-node-1a0a:~$ sudo passwd DevDan
Enter new UNIX password: lfs458
Retype new UNIX password: lfs458
passwd: password updated successfully
```

4. Generate a private key then Certificate Signing Request (CSR) for DevDan.

```
student@lfs458-node-1a0a:~$ openssl genrsa -out DevDan.key 2048
Generating RSA private key, 2048 bit long modulus
.....+++
.....+++
e is 65537 (0x10001)

student@lfs458-node-1a0a:~$ openssl req -new -key DevDan.key \
-out DevDan.csr -subj "/CN=DevDan/O=development"
```

5. Using the 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@lfs458-node-1a0a:~$ 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

Signature ok
subject=/CN=DevDan/O=development
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@lfs458-node-1a0a:~$ kubectl config set-credentials DevDan \
  --client-certificate=/home/student/DevDan.crt \
  --client-key=/home/student/DevDan.key
User "DevDan" set.
```

7. View the update to your credentials file. Use **diff** to compare against the copy we made earlier.

```
student@lfs458-node-1a0a:~$ diff cluster-api-config .kube/config
16a,19d15
> - name: DevDan
>   user:
>     as-user-extra: {}
>     client-certificate: /home/student/DevDan.crt
>     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@lfs458-node-1a0a:~$ kubectl config set-context DevDan-context \
  --cluster=kubernetes \
  --namespace=development \
  --user=DevDan
Context "DevDan-context" created.
```

9. Attempt to view the Pods inside the DevDan-context. Be aware you will get an error.

```
student@lfs458-node-1a0a:~$ kubectl --context=DevDan-context get pods
Error from server (Forbidden): pods is forbidden: User "DevDan"
cannot list pods in the namespace "development"
```

10. Verify the context has been properly set.

```
student@lfs458-node-1a0a:~$ kubectl config get-contexts
CURRENT  NAME             CLUSTER    AUTHINFO    NAMESPACE
*        DevDan-context   kubernetes DevDan       development
*        kubernetes-admin@kubernetes kubernetes kubernetes-admin
```

11. Again check the recent changes to the cluster access config file.

```
student@lfs458-node-1a0a:~$ diff cluster-api-config .kube/config
<output_omitted>
```

12. We will now create a YAML file to associate RBAC rights to a particular namespace and Role.

```
student@lfs458-node-1a0a:~$ vim role-dev.yaml
```

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@lfs458-node-1a0a:~$ kubectl create -f role-dev.yaml
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@lfs458-node-1a0a:~$ vim rolebind.yaml
```

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@lfs458-node-1a0a:~$ kubectl apply -f rolebind.yaml
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@lfs458-node-1a0a:~$ kubectl --context=DevDan-context get pods
No resources found in development namespace.
```

16. Create a new pod, verify it exists, then delete it.

```
student@lfs458-node-1a0a:~$ kubectl --context=DevDan-context \
  create deployment nginx --image=nginx
deployment.apps/nginx created

student@lfs458-node-1a0a:~$ kubectl --context=DevDan-context get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-7c87f569d-7gb9k              1/1     Running   0           5s

student@lfs458-node-1a0a:~$ kubectl --context=DevDan-context delete \
  deploy nginx
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@lfs458-node-1a0a:~$ cp role-dev.yaml role-prod.yaml

student@lfs458-node-1a0a:~$ vim role-prod.yaml
```




role-prod.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@lfs458-node-1a0a:~$ cp rolebind.yaml rolebindprod.yaml
```

```
student@lfs458-node-1a0a:~$ vim rolebindprod.yaml
```



rolebindprod.yaml

```

1 kind: RoleBinding
2 apiVersion: rbac.authorization.k8s.io/v1
3 metadata:
4   name: production-role-binding
5   namespace: production
6 subjects:
7 - kind: User
8   name: DevDan
9   apiGroup: ""
10 roleRef:
11   kind: Role
12   name: dev-prod
13   apiGroup: ""

```

18. Create both new objects.

```
student@lfs458-node-1a0a:~$ kubectl apply -f role-prod.yaml
role.rbac.authorization.k8s.io/dev-prod created
```

```
student@lfs458-node-1a0a:~$ kubectl apply -f rolebindprod.yaml
rolebinding.rbac.authorization.k8s.io/production-role-binding created
```

19. Create the new context for production use.

```
student@lfs458-node-1a0a:~$ kubectl config set-context ProdDan-context \
--cluster=kubernetes \
--namespace=production \
--user=DevDan
Context "ProdDan-context" created.
```

20. Verify that user DevDan can view pods using the new context.

```
student@lfs458-node-1a0a:~$ kubectl --context=ProdDan-context get pods
No resources found in production namespace.
```

21. Try to create a Pod in production. The developer should be Forbidden.

```
student@lfs458-node-1a0a:~$ kubectl --context=ProdDan-context create \
deployment nginx --image=nginx
```

```
Error from server (Forbidden): deployments.apps is forbidden:
User "DevDan" cannot create deployments.apps in the
namespace "production"
```

22. View the details of a role.

```
student@lfs458-node-1a0a:~$ kubectl -n production describe role dev-prod

Name:          dev-prod
Labels:        <none>
Annotations:   kubectl.kubernetes.io/last-applied-configuration=
{"apiVersion":"rbac.authorization.k8s.io/v1","kind":"Role",
,"metadata":{"annotations":{},"name":"dev-prod","namespace":
"production"},"rules":[{"api...
PolicyRule:
  Resources          Non-Resource URLs  Resource Names      Verbs
  -----
  deployments        []                 []                  [get list watch]
  deployments.apps   []                 []                  [get list watch]
<output_omitted>
```

23. Experiment with other subcommands in both contexts. They should match those listed in the respective roles.

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@lfs458-node-1a0a:~$ sudo grep admission \
/etc/kubernetes/manifests/kube-apiserver.yaml
- --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 as 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
<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
    option tcplog
    mode tcp
....

    errorfile 503 /etc/haproxy/errors/503.http
    errorfile 504 /etc/haproxy/errors/504.http

frontend proxynode           #<-- Add the following lines
    bind *:80
    bind *:6443
    stats uri /proxystats
    default_backend k8sServers

backend k8sServers
    balance roundrobin
    server lfs458-node-1a0a 10.128.0.24:6443 check #<-- Edit these with your IP addresses and host names
#   server lfs458-SecondMaster 10.128.0.30:6443 check
#   server lfs458-ThirdMaster 10.128.0.66:6443 check
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
<output_omitted>
Aug 08 18:43:08 ha-proxy systemd[1]: Starting HAProxy Load Balancer...
Aug 08 18:43:08 ha-proxy systemd[1]: Started HAProxy Load Balancer.
Aug 08 18:43:08 ha-proxy haproxy-systemd-wrapper[13602]: haproxy-systemd-wrapper:
Aug 08 18:43:08 ha-proxy haproxy[13603]: Proxy proxynode started.
Aug 08 18:43:08 ha-proxy haproxy[13603]: Proxy proxynode started.
Aug 08 18:43:08 ha-proxy haproxy[13603]: Proxy k8sServers started.
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@lfs458-node-1a0a:~$ 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://35.238.XX.YY:9999/stats` is an example your IP address would be different. Leave the browser up and refresh as you run following steps.

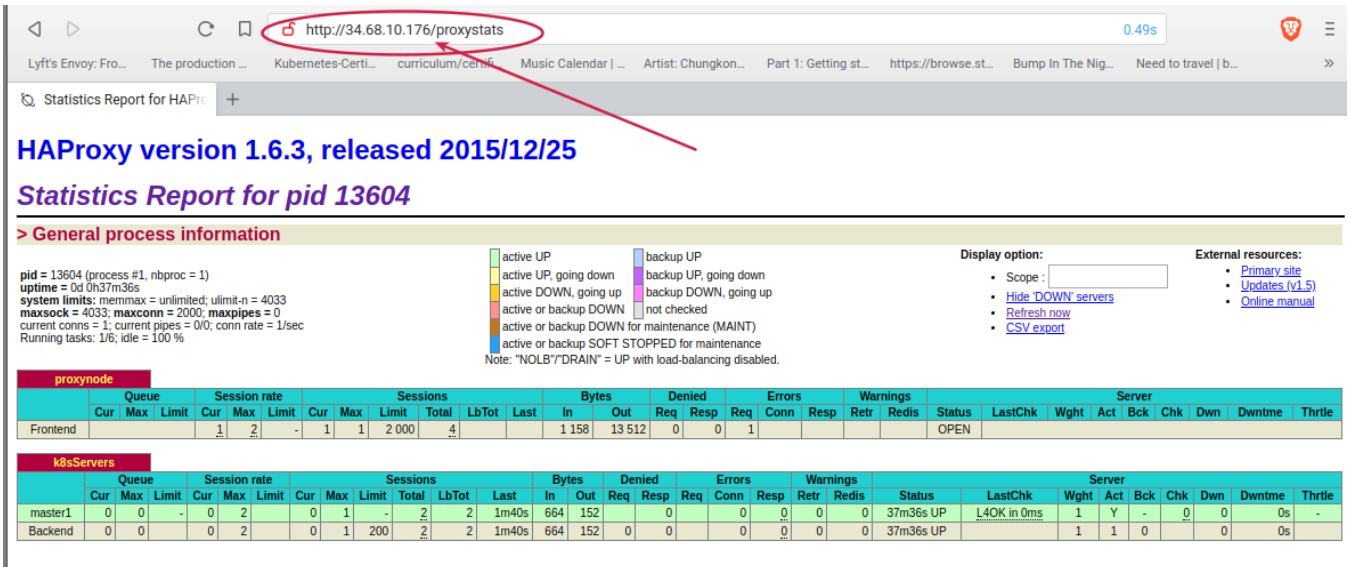


Figure 16.1: Initial HAProxy Status

6. Check the node status from the master node then check the proxy statistics. You should see the byte traffic counter increase.

```
student@lfs458-node-1a0a:~$ kubectl get nodes
```

```
NAME                STATUS    ROLES    AGE    VERSION
lfs458-node-1a0a    Ready    master   2d6h   v1.16.1
lfs458-worker        Ready    <none>   2d3h   v1.16.1
```

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
root@SecondMaster:~$ apt-get install -y docker.io
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=1.16.1-00 kubelet=1.16.1-00 kubectl=1.16.1-00
root@SecondMaster:~$ exit

```

2. 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@lfs458-node-1a0a:~$ sudo vim /etc/hosts
10.128.0.64 k8smaster
#10.128.0.24 k8smaster
127.0.0.1 localhost
....

```

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
jasg79.fdh4p2791320cz1g

```

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/^.* //'
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
[upload-certs] Storing the certificates in Secret "kubeadm-certs" in the "kube-system" Namespace
[upload-certs] Using certificate key:
5610b6f73593049acddee6b59994360aa4441be0c0d9277c76705d129ba18d65

```

6. On the **second master** use the previous output to build a `kubeadm join` command.

```

student@SecondMaster:~$ sudo kubeadm join k8smaster:6443 \
--token jasg79.fdh4p2791320cz1g \
--discovery-token-ca-cert-hash sha256:f62bf97d4fba6876e4c3ff645df3fca969c06169dee3865aab9d0bca8ec9f8cd \
--control-plane --certificate-key \
5610b6f73593049acddee6b59994360aa4441be0c0d9277c76705d129ba18d65
[preflight] Running pre-flight checks
[WARNING IsDockerSystemdCheck]: detected "cgroups" as the Docker cgroup driver. The recommended driver \
is "systemd". Please follow the guide at https://kubernetes.io/docs/setup/cri/
<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@lfs458-node-1a0a:~$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
lfs458-node-1a0a	Ready	master	2d6h	v1.16.1
lfs458-worker	Ready	<none>	2d3h	v1.16.1
lfs458-SecondMaster	Ready	master	10m	v1.16.1

8. Copy and paste the **kubeadm join** command to the third master. Then check that the third master has been added.

```
student@lfs458-node-1a0a:~$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
lfs458-node-1a0a	Ready	master	2d6h	v1.16.1
lfs458-worker	Ready	<none>	2d3h	v1.16.1
lfs458-SecondMaster	Ready	master	10m	v1.16.1
lfs458-ThirdMaster	Ready	master	3m4s	v1.16.1

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
```

```
....
backend k8sServers
    balance roundrobin
    server lfs458-node-1a0a 10.128.0.24:6443 check
    server lfs458-SecondMaster 10.128.0.30:6443 check #<-- Edit/Uncomment these lines
    server lfs458-ThirdMaster 10.128.0.66:6443 check #<--
....

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		Denied		
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	
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@lfs458-node-1a0a:~$ kubectl -n kube-system get pods |grep etcd
etcd-lfs458-node-1a0a          1/1      Running   0          2d12h
etcd-lfs458-SecondMaster      1/1      Running   0          22m
etcd-lfs458-ThirdMaster       1/1      Running   0          18m

student@lfs458-node-1a0a:~$ kubectl -n kube-system logs -f etcd-lfs458-ThirdMaster
....
2019-08-09 01:58:03.768858 I | mvcc: store.index: compact 300473
2019-08-09 01:58:03.770773 I | mvcc: finished scheduled compaction at 300473 (took 1.286565ms)
2019-08-09 02:03:03.766253 I | mvcc: store.index: compact 301003
2019-08-09 02:03:03.767582 I | mvcc: finished scheduled compaction at 301003 (took 995.775µs)
2019-08-09 02:08:03.785807 I | mvcc: store.index: compact 301533
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.

```
student@lfs458-node-1a0a:~$ kubectl -n kube-system exec -it etcd-lfs458-node-1a0a -- /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
```

```
+-----+-----+-----+-----+-----+-----+-----+
| ENDPOINT | ID | VERSION | DB SIZE | IS LEADER | RAFT TERM | RAFT INDEX |
+-----+-----+-----+-----+-----+-----+-----+
| 10.128.0.66:2379 | 2331065cd4fb02ff | 3.3.10 | 24 MB | true | 11 | 392573 |
```




```
| 10.128.0.24:2379 | d2620a7d27a9b449 | 3.3.10 | 24 MB | false | 11 | 392573 |
| 10.128.0.30:2379 | ef44cc541c5f37c7 | 3.3.10 | 24 MB | false | 11 | 392573 |
+-----+-----+-----+-----+-----+-----+-----+
```

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. Shut down the docker service on node which has IS LEADER set to true.

```
student@lfs458-node-1a0a:~$ sudo systemctl stop docker.service
```

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@lfs458-node-1a0a:~$ kubectl -n kube-system logs -f etcd-lfs458-ThirdMaster
```

```
....
2019-08-09 02:11:39.569827 I | raft: 2331065cd4fb02ff [term: 9] received a MsgVote message with higher \
term from ef44cc541c5f37c7 [term: 10]
2019-08-09 02:11:39.570130 I | raft: 2331065cd4fb02ff became follower at term 10
2019-08-09 02:11:39.570148 I | raft: 2331065cd4fb02ff [logterm: 9, index: 355240, vote: 0] cast MsgVote \
for ef44cc541c5f37c7 [logterm: 9, index: 355240] at term 10
2019-08-09 02:11:39.570155 I | raft: raft.node: 2331065cd4fb02ff lost leader d2620a7d27a9b449 at term 10
2019-08-09 02:11:39.572242 I | raft: raft.node: 2331065cd4fb02ff elected leader ef44cc541c5f37c7 at \
term 10
2019-08-09 02:11:39.682319 W | rafthttp: lost the TCP streaming connection with peer d2620a7d27a9b449 \
(stream Message reader)
2019-08-09 02:11:39.682635 W | rafthttp: lost the TCP streaming connection with peer d2620a7d27a9b449 \
(stream MsgApp v2 reader)
2019-08-09 02:11:39.706068 E | rafthttp: failed to dial d2620a7d27a9b449 on stream MsgApp v2 \
(peer d2620a7d27a9b449 failed to find local node 2331065cd4fb02ff)
2019-08-09 02:11:39.706328 I | rafthttp: peer d2620a7d27a9b449 became inactive (message send to peer failed)
....
```

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	Wght		
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr				Redis	
master1	0	0	-	0	22		0	23	-	173	129	12m18s	11 110 233	62 695 354	0	0	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	0	0	4h15m UP	L4OK in 0ms		
master3	0	0	-	0	23		5	22	-	128	128	12m23s	362 790	6 078 463	0	0	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	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
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

```
/ # ETCDCCTL_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
```

Failed to get the status of endpoint 10.128.0.66:2379 (context deadline exceeded)

ENDPOINT	ID	VERSION	DB SIZE	IS LEADER	RAFT TERM	RAFT INDEX
10.128.0.24:2379	d2620a7d27a9b449	3.3.10	24 MB	true	12	395729
10.128.0.30:2379	ef44cc541c5f37c7	3.3.10	24 MB	false	12	395729

- Turn the docker service back on. You should see the peer become active and establish a connection.

```
student@lfs458-node-1a0a:~$ sudo systemctl start docker.service
```

```
student@lfs458-node-1a0a:~$ kubectl -n kube-system logs -f etcd-lfs458-ThirdMaster
```

```
....
2019-08-09 02:45:11.337669 I | rafthttp: peer d2620a7d27a9b449 became active
2019-08-09 02:45:11.337710 I | rafthttp: established a TCP streaming connection with peer\
d2620a7d27a9b449 (stream MsgApp v2 reader)
....
```

- 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.