LFS258

Kubernetes Fundamentals

Version 2023-09-14

THE LINUX FOUNDATION | Training & Certification

Version 2023-09-14

Published by:

the **Linux Foundation**
https://www.linuxfoundation.org

Nondisclosure of Confidential Information

“Confidential Information” shall not include any of the following, even if marked confidential or proprietary: (a) information that relates to the code base of any open source or open standards project (collectively, “Open Project”), including any existing or future contribution thereto; (b) information generally relating or pertaining to the formation or operation of any Open Project; or (c) information relating to general business matters involving any Open Project.

This course does not include confidential information, nor should any confidential information be divulged in class.

Contents

List of Figures

Chapter 1

Introduction

1.1 Labs

Exercise 1.1: Configuring the System for sudo

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

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

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

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

```
$ sudo ls
```

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

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

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

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

if the user is `student`.

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

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

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

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

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

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

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

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

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

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

Chapter 2

Basics of Kubernetes

2.1 Labs

Exercise 2.1: View Online Resources

Visit kubernetes.io

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

1. Open a browser and visit the https://kubernetes.io/ website.
2. In the upper right hand corner, use the drop down to view the versions available. It will say something like `v1.25`.
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.

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: Digital Ocean, GCP, AWS, VirtualBox, etc

The labs were written using **Ubuntu 20.04** instances running on **G**oogle **C**loud **P**latform (**GCP**). They have been written to be vendor-agnostic so could run on AWS, local hardware, or inside of virtualization to give you the most flexibility and options. Each platform will have different access methods and considerations. As of v1.21.0 the minimum (as in barely works) size for **VirtualBox** is 3vCPU/4G memory/5G minimal OS for `cp` and 1vCPU/2G memory/5G minimal OS for worker node. Most other providers work with 2CPU/7.5G.

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

Very Important

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

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

YAML files and White Space

Various exercises will use YAML files, which are included in the text. You are encouraged to write some of the files as time permits, 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 mentioned in labs have also been made available as a compressed **tar** file. You can view the resources by navigating to this URL:

https://cm.lf.training/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://cm.lf.training/LFS258/LFS258_V2023-09-14_SOLUTIONS.tar.xz \
              --user=LFtraining --password=Penguin2014

$ tar -xvf LFS258_V2023-09-14_SOLUTIONS.tar.xz
```

(**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!)

Install Kubernetes

Log into your control plane (cp) and worker 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`. Create a non-root user if one is not present. The IP used in the example will be different than the one you will use. You may need to adjust the access mode of your pem or ppk key. The example shows how a Mac or Linux system would change mode. Windows may have a similar process.

```
[student@laptop ~]$ chmod 400 LFS258.pem
[student@laptop ~]$ ssh -i LFS258.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. Use the **wget** command above to download and extract the course tarball to your node. Again copy and paste won't always paste the underscore characters.

3. You are encouraged to type out commands, if using a PDF or eLearning, instead of copy and paste. By typing the commands you have a better chance to remember both the command and the concept. There are a few exceptions, such as when a long hash or output is much easier to copy over, and does not offer a learning opportunity.

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

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

root@cp:~# 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>
```

5. Install a text editor like **nano** (an easy to use editor), **vim**, or **emacs**. Any will do, the labs use a popular option, **vim**.

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

```
<output-omitted>
```

6. The main choices for a container environment are **containerd**, **cri-o**, and **Docker** on older clusters. We suggest **containerd** for class, as it is easy to deploy and commonly used by cloud providers.

   Please note, install one engine only. If more than one are installed the **kubeadm** init process search pattern will use Docker at the moment. Also be aware that engines other than **containerd** may show different output on some commands.

7. There are several packages we should install to ensure we have all dependencies take care of. Please note the backslash is not necessary and can be removed if typing on a single line.

```
root@cp:~# apt install curl apt-transport-https vim git wget  \
software-properties-common lsb-release ca-certificates  -y
```

```
<output-omitted>
```

8. Disable swap if not already done. Cloud providers disable swap on their images.

```
root@cp:~#  swapoff -a
```

9. Load modules to ensure they are available for following steps.

```
root@cp:~# modprobe overlay
root@cp:~# modprobe br_netfilter
```

10. Update kernel networking to allow necessary traffic. Be aware the shell will add a greater than sign (>) to indicate the command continues after a carriage return.

```
root@cp:~# cat << EOF | tee /etc/sysctl.d/kubernetes.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
EOF
```

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

11. Ensure the changes are used by the current kernel as well

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

```
* Applying /etc/sysctl.d/10-console-messages.conf ...
kernel.printk = 4 4 1 7
* Applying /etc/sysctl.d/10-ipv6-privacy.conf ...
net.ipv6.conf.all.use_tempaddr = 2
net.ipv6.conf.default.use_tempaddr = 2
* Applying /etc/sysctl.d/10-kernel-hardening.conf ...
kernel.kptr_restrict = 1
<output_omitted>
```

12. Install the necessary key for the software to install

```
root@cp:~# sudo mkdir -p /etc/apt/keyrings
root@cp:~# curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
| sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

root@cp:~# echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] \
  https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

13. Install the containerd software.

```
root@cp:~# apt-get update &&  apt-get install containerd.io -y
root@cp:~# containerd config default | tee /etc/containerd/config.toml
root@cp:~# sed -e 's/SystemdCgroup = false/SystemdCgroup = true/g' -i /etc/containerd/config.toml
root@cp:~# systemctl restart containerd
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
<output_omitted>
```

14. 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 20.04` but the `kubernetes-xenial` repo of the software, also include the key word `main`. Note there are four sections to the entry.

```
root@cp:˜# vim /etc/apt/sources.list.d/kubernetes.list
```

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

15. 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@cp:˜# curl -s \
   https://packages.cloud.google.com/apt/doc/apt-key.gpg \
   | apt-key add -
```

```
OK
```

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

```
root@cp:˜# apt-get update
```

```
<output-omitted>
```

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

```
root@cp:˜# apt-get install -y kubeadm=1.27.1-00 kubelet=1.27.1-00 kubectl=1.27.1-00
```

```
<output-omitted>
```

```
root@cp:˜# apt-mark hold kubelet kubeadm kubectl
```

```
kubelet set on hold.
kubeadm set on hold.
kubectl set on hold.
```

18. Find the IP address of the primary interface of the cp server. The example below would be the `ens4` interface and an IP of `10.128.0.3`, yours may be different. There are two ways of looking at your IP addresses.

```
root@cp:˜# hostname -i
```

```
10.128.0.3
```

```
root@cp:˜# 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
....
```

19. Add an local DNS alias for our cp server. Edit the `/etc/hosts` file and add the above IP address and assign a name `k8scp`.

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

```
10.128.0.3 k8scp    #<-- Add this line
127.0.0.1 localhost
....
```

20. Create a configuration file for the cluster. There are many options we could include, and they differ for **containerd**, **Docker**, and **cri-o**. Use the file included in the course tarball. After our cluster is initialized we will view other default values used. Be sure to use the node alias we added to `/etc/hosts`, not the IP so the network certificates will continue to work when we deploy a load balancer in a future lab. The file is also in the course tarball.

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

kubeadm-config.yaml

```yaml
apiVersion: kubeadm.k8s.io/v1beta3
kind: ClusterConfiguration
kubernetesVersion: 1.27.1               #<-- Use the word stable for newest version
controlPlaneEndpoint: "k8scp:6443"      #<-- Use the alias we put in /etc/hosts not the IP
networking:
  podSubnet: 192.168.0.0/16             #<-- Match the IP range from the CNI config file
```

21. Initialize the cp. Scan through the output. 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 **Cilium** has in its configuration file. **Please note:** the output lists several commands which following exercise steps will complete.

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

```
[init] Using Kubernetes version: v1.27.1
[preflight] Running pre-flight checks



<output_omitted>

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

kubeadm join k8scp: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 k8scp:6443 --token vapzqi.et2p9zbkzk29wwth \
 --discovery-token-ca-cert-hash
 ↪  sha256:f62bf97d4fba6876e4c3ff645df3fca969c06169dee3865aab9d0bca8ec9f8cd
```

22. 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@cp:~# exit
```

```
logout
```

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

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

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

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

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

23. 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. We will use **Cilium** as a network plugin which will allow us to use `Network Policies` later in the course. Currently **Cilium** does not deploy using CNI by default.

    Cilium is genereally installed using ”cilium install” or using ”helm install” commands. We have generated the cilium-cni.yaml file using the below commands for your convenience. **Note**: You dont need to execute the commands in this box, they are just for reference.

```
$ helm repo add cilium https://helm.cilium.io/
$ helm repo update
$ helm template cilium cilium/cilium --version 1.14.1 \
  --namespace kube-system > cilium.yaml
```

```
student@cp:~$ find $HOME -name cilium-cni.yaml
student@cp:~$ kubectl apply -f /home/student/LFS258/SOLUTIONS/s_03/cilium-cni.yaml
```

```
serviceaccount/cilium created
serviceaccount/cilium-operator created
secret/cilium-ca created
configmap/cilium-config created
<output_omitted>
```

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

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

<exit and log back in>

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

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

25. 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 `cp`), 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@cp:~$ kubectl des<Tab> n<Tab><Tab> cp<Tab>
```

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

26. Explore the **kubectl help** command. The output has been omitted from commands. Take a moment to review help topics.

```
student@cp:~$ kubectl help
```

```
student@cp:~$ kubectl help create
```

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

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

```
apiVersion: kubeadm.k8s.io/v1beta3
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 **containerd** and Kubernetes software. These are the many, but not all, of the steps we did on the cp node.

This book will use the **worker** prompt for the node being added to help keep track of the proper node for each command. Note that the prompt indicates both the user and system upon which run the command. It can be helpful to change the colors and fonts of your terminal session to keep track of the correct node.

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.

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

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

```
<If asked allow services to restart and keep the local version of software>
```

4. Install the containerd engine, starting with dependent software.

```
root@worker:~#  apt install curl apt-transport-https vim git wget  \
software-properties-common lsb-release ca-certificates  -y
```

```
root@worker:~# swapoff -a

root@worker:~# modprobe overlay

root@worker:~# modprobe br_netfilter
```

```
root@worker:~# cat << EOF | tee /etc/sysctl.d/kubernetes.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
EOF

root@worker:~# sysctl --system

root@worker:~# mkdir -p /etc/apt/keyrings
root@worker:~# curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
| sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

root@worker:~# echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] \
  https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

root@worker:~# apt-get update &&  apt-get install containerd.io -y
root@worker:~# containerd config default | tee /etc/containerd/config.toml
root@worker:~# sed -e 's/SystemdCgroup = false/SystemdCgroup = true/g' -i /etc/containerd/config.toml
root@worker:~# systemctl restart containerd
```

5. Add Kubernetes repo

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

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

6. Get the GPG key for the software

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

7. Update repos then install the Kubernetes software. Be sure to match the version on the cp.

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

8. `root@worker:~# apt-get install -y \`

```
      kubeadm=1.27.1-00 kubelet=1.27.1-00 kubectl=1.27.1-00
```

9. Ensure the version remains if the system is updated.

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

10. Find the IP address of your **cp** 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 cp node IP is `10.128.0.3`.

```
student@cp:~$ hostname -i
```

```
10.128.0.3
```

```
student@cp:~$ 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
```

11. At this point we could copy and paste the **join** command from the cp 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 cp 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@cp:~$ sudo kubeadm token list
```

```
TOKEN                     TTL       EXPIRES                USAGES
   DESCRIPTION                              EXTRA GROUPS
bml44w.3owxl50rrtymamt7   2h        2023-05-27T18:49:41Z   authentication,signing
  <none>                                    system:bootstrappers:kubeadm:default-node-token
```

12. 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@cp:~$ sudo kubeadm token create
```

```
27eee4.6e66ff60318da929
```

13. Create and use a Discovery Token CA Cert Hash created from the cp to ensure the node joins the cluster in a secure manner. Run this on the cp node or wherever you have a copy of the CA file. You will get a long string as output. Also note that a copy and paste from a PDF sometimes has issues with the caret (^) and the single quote (') found at the end of the command.

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

```
6d541678b05652e1fa5d43908e75e67376e994c3483d6683f2a18673e5d2a1b0
```

14. On the **worker node** add a local DNS alias for the cp server. Edit the `/etc/hosts` file and add the cp IP address and assign the name `k8scp`. The entry should be exactly the same as the edit on the cp.

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

10.128.0.3 k8scp     #<-- Add this line
127.0.0.1 localhost
....
```

15. 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 cp server and port 6443. The output of the **kubeadm init** on the cp also has an example to use, should it still be available.

```
root@worker:~# kubeadm join \
     --token 27eee4.6e66ff60318da929 \
     k8scp:6443 \
     --discovery-token-ca-cert-hash \
     sha256:6d541678b05652e1fa5d43908e75e67376e994c3483d6683f2a18673e5d2a1b0
```

```
[preflight] Running pre-flight checks


[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] 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 apiserver 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.
```

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

```
root@worker:~# exit

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

```
The connection to the server localhost:8080 was refused - did you specify the right host or port?
```

```
student@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 may be different, use YOUR `control-plane` name in future commands, if different than the book.

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

```
NAME     STATUS   ROLES           AGE    VERSION
cp       Ready    control-plane   28m    v1.27.1
worker   Ready    <none>          50s    v1.27.1
```

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 cp 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@cp:~$ kubectl describe node cp
```

```
Name:               cp
Roles:              control-plane,master
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/arch=amd64
                    kubernetes.io/hostname=cp
                    kubernetes.io/os=linux
                    node-role.kubernetes.io/control-plane=
Annotations:        kubeadm.alpha.kubernetes.io/cri-socket: /var/run/dockershim.sock
                    node.alpha.kubernetes.io/ttl: 0
                    volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp:  Tue, 23 Aug 2023 22:04:03 +0000
Taints:             node-role.kubernetes.io/master:NoSchedule
<output_omitted>
```

3. Allow the cp server to run non-infrastructure pods. The cp 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. There may be more than one taint. Keep checking and removing them until all are removed.

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

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

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

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

```
student@cp:~$ kubectl taint nodes --all node-role.kubernetes.io/control-plane-
```

```
node/cp untainted
error: taint "node-role.kubernetes.io/control-plane" not found
```

4. Determine if the DNS and Cilium 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@cp:~$ kubectl get pods --all-namespaces
```

```
NAMESPACE     NAME                                READY   STATUS    RESTARTS   AGE
kube-system   cilium-operator-788c7d7585-tnsph    1/1     Running   0          95m
kube-system   cilium-swjsj                        1/1     Running   0          95m
kube-system   coredns-5d78c9869d-dwds8            1/1     Running   0          100m
kube-system   coredns-5d78c9869d-t24p5            1/1     Running   0          100m

<output_omitted>
```

5. **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@cp:~$ kubectl get pods --all-namespaces
```

```
NAMESPACE     NAME                        READY   STATUS              RESTARTS   AGE
kube-system   cilium-swjsj                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@cp:~$ kubectl -n kube-system delete \
    pod coredns-576cbf47c7-vq5dz coredns-576cbf47c7-rn6v4
```

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

6. 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@cp:~$ 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>
```

7. Containerd may still be using an out of date notation for the `runtime-endpoint`. You may see errors about an undeclared resource type such as `unix//:`. We will update the **crictl** configuration. There are many possible configuration options. We will set one, and view the configuration file that is created.

```
student@cp:~$ sudo crictl config --set \
runtime-endpoint=unix:///run/containerd/containerd.sock \
--set image-endpoint=unix:///run/containerd/containerd.sock
```

```
student@cp:~$ sudo cat /etc/crictl.yaml
```

```
runtime-endpoint: "unix:///run/containerd/containerd.sock"
image-endpoint: "unix:///run/containerd/containerd.sock"
timeout: 0
debug: false
pull-image-on-create: false
disable-pull-on-run: false
```

Exercise 3.4: Deploy A Simple Application

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

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

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

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

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

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

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

```
student@cp:~$ kubectl describe deployment nginx
```

```
Name:                   nginx
Namespace:              default
CreationTimestamp:      Wed, 23 Aug 2023 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>
```

3. View the basic steps the cluster took in order to pull and deploy the new application. You should see several lines of output. The first column shows the age of each message, note that due to JSON lack of order the time `LAST SEEN` time does not print out chronologically. Eventually older messages will be removed.

```
student@cp:~$ kubectl get events
```

```
<output_omitted>
```

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

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

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  creationTimestamp: 2023-08-24T18:21:25Z
<output_omitted>
```

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

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

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

```
<Remove the lines mentioned above>
```

6. Delete the existing deployment.

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

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

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

```
student@cp:~$ 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 creation time stamp, resource version and unique ID we had deleted are in the new file. These are generated for each resource we create, so we may need to delete them from yaml files to avoid conflicts or false information. You may notice some time stamp differences as well. The `status` should not be hard-coded either.

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

```
student@cp:~$ 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 the same essential values.

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

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
```

```yaml
  labels:
    app: two
  name: two
spec:
<output_omitted>
```

```
student@cp:~$ 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.

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

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  creationTimestamp: null
  generation: 1
  labels:
    run: nginx
<output_omitted>
```

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

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

```json
{
    "apiVersion": "apps/v1",
    "kind": "Deployment",
    "metadata": {
        "annotations": {
            "deployment.kubernetes.io/revision": "1"
        },
<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@cp:~$ 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@cp:~$ 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@cp:~$ vim first.yaml
    ```

    first.yaml

    ```yaml
    ....
      spec:
        containers:
        - image: nginx
          imagePullPolicy: Always
          name: nginx
          ports:                              # Add these
          - containerPort: 80                 # three
            protocol: TCP                     # lines
          resources: {}
    ....
    ```

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

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

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

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

    ```
    student@cp:~$ 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@cp:~$ 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. Cilium 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@cp:~$ kubectl get svc nginx
    ```

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

    ```
    student@cp:~$ 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**, which you may need to install using **apt-get install**, 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 `cili` 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@cp:~$ kubectl describe pod nginx-7cbc4b4d9c-d27xw \
                  | grep Node:
```

```
Node:   worker/10.128.0.5
```

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

```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on cilium_vxlan, link-type EN10MB (Ethernet), capture size 262144 bytes
<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@cp:~$ curl 10.100.61.122:80
```

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

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

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

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

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

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

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

```
student@cp:~$ 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@cp:~$ 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 cp and the 192.168.1 addresses are probably on the worker

```
student@cp:~$ kubectl get pod -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@cp:~$ kubectl delete pod 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. Also note we are using a short name for the object.

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

```
service/nginx exposed
```

```
student@cp:~$ 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
```

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

```
student@cp:~$ curl ifconfig.io
```

```
54.214.214.156
```

Figure 3.1: **External Access via Browser**

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

```
student@cp:˜$ kubectl scale deployment nginx --replicas=0
```

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

```
student@cp:˜$ kubectl get po
```

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

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

```
student@cp:˜$ kubectl scale deployment nginx --replicas=2
```

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

```
student@cp:˜$ kubectl get po
```

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

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

```
student@cp:˜$ kubectl delete deployments nginx
```

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

```
student@cp:˜$ kubectl delete ep nginx
```

```
endpoints "nginx" deleted
```

```
student@cp:˜$ kubectl delete svc nginx
```

```
service "nginx" deleted
```

Chapter 4

Kubernetes Architecture

4.1 Labs

Exercise 4.1: Basic Node Maintenance

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

Backup The etcd Database

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

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

```
student@cp:~$ sudo grep data-dir /etc/kubernetes/manifests/etcd.yaml
```

```
    - --data-dir=/var/lib/etcd
```

2. Log into the **etcd** container and look at the options **etcdctl** provides. Use tab to complete the container name, which has the node name appended to it.

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

On Container

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

```
# etcdctl -h
```

```
NAME:
        etcdctl - A simple command line client for etcd3.

USAGE:
        etcdctl [flags]
<output_omitted>
```

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

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

```
# echo *
```

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

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

```
# exit
```

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

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

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

4. Determine how many databases are part of the cluster. Three and five are common in a production environment to provide 50%+1 for quorum. In our current exercise environment we will only see one database. Remember you can use `up-arrow` to return to the previous command and edit the command without having to type the whole command again.

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

```
fb50b7ddbf4930ba, started, cp, https://10.128.0.35:2380,
https://10.128.0.35:2379, false
```

5. You can also view the status of the cluster in a table format, among others passed with the **-w** option. Again, up-arrow allows you to edit just the last part of the long string easily.

```
student@cp:~$ kubectl -n kube-system exec -it etcd-cp -- sh \
 -c "ETCDCTL_API=3 \
 ETCDCTL_CACERT=/etc/kubernetes/pki/etcd/ca.crt \
```

```
 ETCDCTL_CERT=/etc/kubernetes/pki/etcd/server.crt \
 ETCDCTL_KEY=/etc/kubernetes/pki/etcd/server.key \
 etcdctl --endpoints=https://127.0.0.1:2379 member list -w table"
```

```
+------------------+---------+-------+--------------------------+--------------------------+------------+
|        ID        | STATUS  | NAME  |        PEER ADDRS        |       CLIENT ADDRS       | IS
↪  LEARNER |
+------------------+---------+-------+--------------------------+--------------------------+------------+
| 802d78549985d5a8 | started | cp | https://10.128.0.15:2380 | https://10.128.0.15:2379 |
↪  false |
+------------------+---------+-------+--------------------------+--------------------------+------------+
```

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

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

```
{"level":"info","ts":1598380941.6584022,"caller":"snapshot/v3_snapshot.go:110","
msg":"created temporary db file","path":"/var/lib/etcd/snapshot.db.part"}
{"level":"warn","ts":"2023-02-25T18:42:21.671Z","caller":"clientv3/retry_interceptor.go
:116","msg":"retry stream intercept"}
{"level":"info","ts":1598380941.6736135,"caller":"snapshot/v3_snapshot.go:121","
msg":"fetching snapshot","endpoint":"https://127.0.0.1:2379"}
{"level":"info","ts":1598380941.7519674,"caller":"snapshot/v3_snapshot.go:134","
msg":"fetched snapshot","endpoint":"https://127.0.0.1:2379","took":0.093466104}
{"level":"info","ts":1598380941.7521122,"caller":"snapshot/v3_snapshot.go:143","
msg":"saved","path":"/var/lib/etcd/snapshot.db"}
Snapshot saved at /var/lib/etcd/snapshot.db
```

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

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

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

8. Backup the snapshot as well as other information used to create the cluster both locally as well as another system in case the node becomes unavailable. Remember to create snapshots on a regular basis, perhaps using a cronjob to ensure a timely restore. When using the `snapshot restore` it's important the database not be in use. An HA cluster would remove and replace the control plane node, and not need a restore.

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

9. Any mistakes during restore may render the cluster unusable. Instead of issues, and having to rebuild the cluster, please attempt a database restore after the final lab exercise of the course. More on the restore process can be found here: https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/#restoring-an-etcd-cluster

Upgrade the Cluster

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

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

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

2. View the available packages. The list will be long, you may have to scroll back up to the top to find a recent version. Choose the next full version from what you installed. For example if you used `1.24.1` to initialize the cluster in a previous lab, you would choose `1.25.1`. If you used `1.27.1`, you would look for `1.28.1`. If you initialized using 1.29.1 you would look for 1.30.1 and so forth. The use of the term "madison" is from the Debian tool called madison.

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

```
kubeadm |  1.28.1-00 | http://apt.kubernetes.io kubernetes-xenial/main amd64 Packages
   kubeadm |  1.28.0-00 | http://apt.kubernetes.io kubernetes-xenial/main amd64 Packages
   kubeadm |  1.27.5-00 | http://apt.kubernetes.io kubernetes-xenial/main amd64 Packages
   kubeadm |  1.27.4-00 | http://apt.kubernetes.io kubernetes-xenial/main amd64 Packages
   kubeadm |  1.27.3-00 | http://apt.kubernetes.io kubernetes-xenial/main amd64 Packages
   kubeadm |  1.27.2-00 | http://apt.kubernetes.io kubernetes-xenial/main amd64 Packages
   kubeadm |  1.27.1-00 | http://apt.kubernetes.io kubernetes-xenial/main amd64 Packages
<output_omitted>
```

3. Remove the hold on **kubeadm** and update the package. Remember to update to the next major release's update 1.

```
student@cp:~$ sudo apt-mark unhold kubeadm
```

```
Canceled hold on kubeadm.
```

```
student@cp:~$ sudo apt-get install -y kubeadm=1.28.1-00
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
<output_omitted>
```

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

```
student@cp:~$ sudo apt-mark hold kubeadm
```

```
kubeadm set on hold.
```

5. Verify the version of **Kubeadm**. It should indicate the new version you just installed.

```
student@cp:~$ sudo kubeadm version
```

```
kubeadm version: &version.Info{Major:"1", Minor:"28", GitVersion:"v1.28.1",
GitCommit:"8dc49c4b984b897d423aab4971090e1879eb4f23", GitTreeState:"clean",
BuildDate:"2023-08-24T11:21:51Z", GoVersion:"go1.20.7", Compiler:"gc",
Platform:"linux/amd64"}
```

6. To prepare the cp node for update we first need to evict as many pods as possible. The nature of daemonsets is to have them on every node, and some such as `Cilium` must remain. Change the node name to your node's name, and add the option to ignore the daemonsets.

```
student@cp:~$ kubectl drain cp --ignore-daemonsets
```

```
node/cp cordoned
Warning: ignoring DaemonSet-managed Pods: kube-system/cilium-5tv9d, kube-system/kube-proxy-8x9c5
evicting pod kube-system/coredns-5d78c9869d-z5ngb
```

```
evicting pod kube-system/cilium-operator-788c7d7585-wnb5b
evicting pod kube-system/coredns-5d78c9869d-4h2bs
pod/cilium-operator-788c7d7585-wnb5b evicted
pod/coredns-5d78c9869d-z5ngb evicted
pod/coredns-5d78c9869d-4h2bs evicted
node/cp drained
```

7. Use the **upgrade plan** argument to check the existing cluster and then update the software. You may notice that there are versions available later than the `.1` update in use. If you initialized the cluster in a previous lab using 1.25.1, use upgrade to 1.26.1. If you initialized using 1.26.1, upgrade 1.27.1, etc. Read through the output and get a feel for what would be changed in an upgrade.

```
student@cp:~$ sudo kubeadm upgrade plan
```

```
[upgrade/config] Making sure the configuration is correct:
[upgrade/config] Reading configuration from the cluster...
[upgrade/config] FYI: You can look at this config file with 'kubectl -n kube-system get cm
↪ kubeadm-config -o yaml'
[preflight] Running pre-flight checks.
[upgrade] Running cluster health checks
[upgrade] Fetching available versions to upgrade to
[upgrade/versions] Cluster version: v1.27.5
[upgrade/versions] kubeadm version: v1.28.1
[upgrade/versions] Target version: v1.28.1
[upgrade/versions] Latest version in the v1.27 series: v1.27.5

<output_omitted>
```

8. We are now ready to actually upgrade the software. There will be a lot of output. Be aware the command will ask if you want to proceed with the upgrade, answer **y** for yes. Take a moment and look for any errors or suggestions, such as upgrading the version of etcd, or some other package. Again, Use the next minor release, update 1 from the version used previous lab which initialized the cluster. The process will take several minutes to complete.

```
student@cp:~$ sudo kubeadm upgrade apply v1.28.1
```

```
[upgrade/config] Making sure the configuration is correct:
[upgrade/config] Reading configuration from the cluster...
[upgrade/config] FYI: You can look at this config file with 'kubectl -n kube-system get cm
↪ kubeadm-config -o yaml'
[preflight] Running pre-flight checks.
[upgrade] Running cluster health checks
[upgrade/version] You have chosen to change the cluster version to "v1.28.1"
[upgrade/versions] Cluster version: v1.27.5
[upgrade/versions] kubeadm version: v1.28.1
[upgrade] Are you sure you want to proceed? [y/N]: y
[upgrade/prepull] Pulling images required for setting up a Kubernetes cluster
[upgrade/prepull] This might take a minute or two, depending on the speed of your internet
↪ connection
[upgrade/prepull] You can also perform this action in beforehand using 'kubeadm config images
↪ pull'
[upgrade/apply] Upgrading your Static Pod-hosted control plane to version "v1.28.1" (timeout:
↪ 5m0s)...
<output_omitted>
```

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

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

```
NAME     STATUS                     ROLES           AGE    VERSION
cp       Ready,SchedulingDisabled   control-plane   109m   v1.27.1
worker   Ready                      <none>          61m    v1.27.1
```


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

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

```
Canceled hold on kubelet.
Canceled hold on kubectl.
```

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

```
student@cp:~$ sudo apt-get install -y kubelet=1.28.1-00 kubectl=1.28.1-00
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
<output_omitted>
```

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

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

```
kubelet set on hold.
kubectl set on hold.
```

13. Restart the daemons.

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

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

14. Verify the cp node has been updated to the new version. Then update other cp nodes, if you should have them, using the same process except **sudo kubeadm upgrade node** instead of **sudo kubeadm upgrade apply**.

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

```
NAME      STATUS                     ROLES           AGE    VERSION
cp        Ready,SchedulingDisabled   control-plane   113m   v1.28.1
worker    Ready                      <none>          65m    v1.27.1
```

15. Now make the cp available for the scheduler, again change the name to match the cluster node name on your control plane.

```
student@cp:~$ kubectl uncordon cp
```

```
node/cp uncordoned
```

16. Verify the cp now shows a `Ready` status.

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

```
NAME      STATUS     ROLES                 AGE    VERSION
cp        Ready      control-plane         114m   v1.28.1
worker    Ready      <none>                66m    v1.27.1
```

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

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

```
Canceled hold on kubeadm.
```

18. Update the **kubeadm** package to the same version as the cp node.

```
student@worker:~$ sudo apt-get update && sudo apt-get install -y kubeadm=1.28.1-00
```

```
<output_omitted>
Setting up kubeadm (1.28.1-00) ...
```

19. Hold the package again.

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

```
kubeadm set on hold.
```

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

```
student@cp:~$ kubectl drain worker --ignore-daemonsets
```

```
node/worker cordoned
Warning: ignoring DaemonSet-managed Pods: kube-system/cilium-gzdk6, kube-system/kube-proxy-lpsmq
evicting pod kube-system/cilium-operator-788c7d7585-hc9wf
evicting pod kube-system/coredns-5d78c9869d-h4p7v
evicting pod kube-system/coredns-5d78c9869d-d4nv8
pod/cilium-operator-788c7d7585-hc9wf evicted
pod/coredns-5d78c9869d-h4p7v evicted
pod/coredns-5d78c9869d-d4nv8 evicted
node/worker drained
```

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

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

```
[upgrade] Reading configuration from the cluster...
[upgrade] FYI: You can look at this config file with 'kubectl -n kube-system get cm
kubeadm-config -o yaml'
[preflight] Running pre-flight checks
[preflight] Skipping prepull. Not a control plane node.
[upgrade] Skipping phase. Not a control plane node.
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[upgrade] The configuration for this node was successfully updated!
[upgrade] Now you should go ahead and upgrade the kubelet package using your package manager.
```

22. Remove the hold on the software then update to the same version as set on the cp.

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

```
Canceled hold on kubelet.
Canceled hold on kubectl.
```

```
student@worker:~$ sudo apt-get install -y kubelet=1.28.1-00 kubectl=1.28.1-00
```

```
Reading package lists... Done
Building dependency tree
<output_omitted>
Setting up kubelet (1.28.1-00) ...
Setting up kubectl (1.28.1-00) ...
```

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

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

```
kubelet set on hold.
kubectl set on hold.
```

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

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

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

25. Return to the cp node. View the status of the nodes. Notice the worker status.

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

```
NAME      STATUS                       ROLES           AGE    VERSION
cp        Ready                        control-plane   118m   v1.28.1
worker    Ready,SchedulingDisabled     <none>          70m    v1.28.1
```

26. Allow pods to be deployed to the worker node. Remember to use YOUR worker name. `TAB` can be helpful to enter the name if command line completion is enabled.

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

```
node/worker uncordoned
```

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

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

```
NAME     STATUS   ROLES           AGE    VERSION
cp       Ready    control-plane   119m   v1.28.1
worker   Ready    <none>          71m    v1.28.1
```

Exercise 4.2: Working with CPU and Memory Constraints

Overview

We will continue working with our cluster, which we built in the previous lab. We will work with `resource limits`, more with `namespaces` and then a complex `deployment` which you can explore to further understand the architecture and relationships.

Use **SSH** or **PuTTY** to connect to the nodes you installed in the previous exercise. We will deploy an application called **stress** inside a container, and then use `resource limits` to constrain the resources the application has access to use.

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

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

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

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

```
Name:                   hog
Namespace:              default
CreationTimestamp:      Thu, 23 Aug 2023 10:15:53 +0000
Labels:                 app=hog
Annotations:            deployment.kubernetes.io/revision: 1
<output_omitted>
```

```
student@cp:~$ 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.

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

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

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

hog.yaml

```yaml
....
       imagePullPolicy: Always
        name: hog
        resources:                    # Edit to remove {}
          limits:                     # Add these 4 lines
            memory: "4Gi"
          requests:
            memory: "2500Mi"
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
....
```

5. Replace the deployment using the newly edited file.

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

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

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

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

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

```
student@cp:~$ 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 cp and second nodes. Run **top** to view resource usage. You should not see unusual resource usage at this point. The **containerd** and **top** processes should be using about the same amount of resources. The **stress** command should not be using enough resources to show up. Use the **kubectl get events** to see if any pods are evicted when too many resources are in use.

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@cp:~$ vim hog.yaml
```

hog.yaml

```yaml
....
        resources:
          limits:
            cpu: "1"
            memory: "4Gi"
          requests:
            cpu: "0.5"
            memory: "500Mi"
        args:
        - -cpus
        - "2"
        - -mem-total
        - "950Mi"
        - -mem-alloc-size
        - "100Mi"
        - -mem-alloc-sleep
        - "1s"
....
```

10. Delete and recreate the deployment. You should see increased CPU usage almost immediately and memory allocation happen in 100M chunks, allocated to the **stress** program via the running **top** command. Check both nodes as the

container could deployed to either. Be aware that nodes with a small amount of memory or CPU may encounter issues. Symptoms include cp node infrastructure pods failing. Adjust the amount of resources used to allow standard pods to run without error.

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

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

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

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

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

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

```
student@cp:~$ 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 4.3: Resource Limits for a Namespace

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

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

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

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

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

YAML low-resource-range.yaml

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: low-resource-range
spec:
  limits:
  - default:
      cpu: 1
      memory: 500Mi
    defaultRequest:
      cpu: 0.5
      memory: 100Mi
    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@cp:~$ 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@cp:~$ kubectl get LimitRange
```

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

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

```
NAMESPACE         NAME                 CREATED AT
low-usage-limit   low-resource-range   2023-08-23T10:23:57Z
```

5. Create a new deployment in the namespace.

```
student@cp:~$ 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 **Cilium** network policy you may see a couple more than what is listed below.

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

```
NAMESPACE         NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
default           hog                        1/1     1            1           7m57s
kube-system       cilium-operator            1/1     1            1           2d10h
kube-system       coredns                    2/2     2            2           2d10h
low-usage-limit   limited-hog                1/1     1            1           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@cp:~$ 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@cp:~$ 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, if it exists.

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

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

hog2.yaml

```yaml
....
  labels:
    app: hog
  name: hog
  namespace: low-usage-limit     #<<--- Add this line, delete following
  selfLink: /apis/apps/v1/namespaces/default/deployments/hog
spec:
....
```

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@cp:~$ 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 `cilium` deployment has no pods, nor has any requested. Our small cluster does not need to add **Cilium** pods via this autoscaler.

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

```
NAMESPACE         NAME                     READY   UP-TO-DATE   AVAILABLE   AGE
default           hog                      1/1     1            1           24m
kube-system       cilium-operator          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
```

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

```
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
```

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

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

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

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

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

Chapter 5

APIs and Access

5.1 Labs

Exercise 5.1: Configuring TLS Access

Overview

Using the Kubernetes API, **kubectl** makes API calls for you. With the appropriate TLS keys you could run **curl** as well use a **golang** client. Calls to the `kube-apiserver` get or set a PodSpec, or desired state. If the request represents a new state the **Kubernetes Control Plane** will update the cluster until the current state matches the specified state. Some end states may require multiple requests. For example, to delete a `ReplicaSet`, you would first set the number of replicas to zero, then delete the `ReplicaSet`.

An API request must pass information as JSON. **kubectl** converts `.yaml` to JSON when making an API request on your behalf. The API request has many settings, but must include `apiVersion`, `kind` and `metadata`, and `spec` settings to declare what kind of container to deploy. The `spec` fields depend on the object being created.

We will begin by configuring remote access to the `kube-apiserver` then explore more of the API.

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

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

```
<output_omitted>
```

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

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

student@cp:~$ echo $client
```


```
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUM4akNDQWRxZ0F3SUJ
BZ0lJRy9wbC9rWEpNdmd3RFFZSktvWklodmNOQVFFTEJRQXdGVEVUTUJFR0
ExVUUKQXhNS2EzVmlaWEp1WlhSbGN6QWVGdzB4TnpFeU1UTXhOelEyTXpKY
UZ3MHhPREV5TVRNeE56UTJNelJhTURReApGekFWQmd0VkJBb1REbk41YzNS
<output_omitted>
```

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

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

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

```
<output_omitted>
```

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

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

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

```
<output_omitted>
```

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

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

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

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

```
server: https://k8scp: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@cp:~$ curl --cert ./client.pem \
    --key ./client-key.pem \
    --cacert ./ca.pem \
    https://k8scp:6443/api/v1/pods
```

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

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

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

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

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

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

```
student@cp:~$ curl --cert ./client.pem \
   --key ./client-key.pem --cacert ./ca.pem \
   https://k8scp:6443/api/v1/namespaces/default/pods \
   -XPOST -H'Content-Type: application/json' \
   -d@curlpod.json
```

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

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

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

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

Exercise 5.2: Explore API Calls

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

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

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

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

2. Run this command again, preceded by **strace**. You will get a lot of output. Near the end you will note several **openat** functions to a local directory, /home/student/.kube/cache/discovery/k8scp_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@cp:~$ strace kubectl get endpoints
```

```
execve("/usr/bin/kubectl", ["kubectl", "get", "endpoints"], [/*....
....
openat(AT_FDCWD, "/home/student/.kube/cache/discovery/k8scp_6443..
<output_omitted>
```

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

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

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

```
k8scp_6443
```

```
student@cp:~/.kube/cache/discovery$ cd k8scp_6443/
```

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

```
student@cp:~/.kube/cache/discovery/k8scp_6443$ ls
```

```
admissionregistration.k8s.io  certificates.k8s.io           node.k8s.io
apiextensions.k8s.io          coordination.k8s.io           policy
apiregistration.k8s.io        crd.projectcalico.org         rbac.authorization.k8s.io
apps                          discovery.k8s.io              scheduling.k8s.io
authentication.k8s.io         events.k8s.io                 servergroups.json
authorization.k8s.io          extensions                    storage.k8s.io
autoscaling                   flowcontrol.apiserver.k8s.io  v1
batch                         networking.k8s.io
```

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

```
student@cp:./k8scp_6443$ find .
```

```
.
./storage.k8s.io
./storage.k8s.io/v1beta1
./storage.k8s.io/v1beta1/serverresources.json
./storage.k8s.io/v1
./storage.k8s.io/v1/serverresources.json
./rbac.authorization.k8s.io
<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@cp:.$ python3 -m json.tool v1/serverresources.json
```

serverresources.json

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

serverresources.json

```json
....
{
"kind": "Endpoints",
"name": "endpoints",
"namespaced": true,
"shortNames": [
      "ep"
],
"singularName": "",
"verbs": [
   "create",
   "delete",
....
```

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

```
student@cp:.$ 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@cp:.$ 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@cp:$ 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.

```
student@cp:$ 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 cp 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@cp:~$ kubectl config view
   ```

   ```
   apiVersion: v1
   clusters:
   - cluster:
       certificate-authority-data: DATA+OMITTED
       server: https://k8scp:6443
     name: kubernetes
   <output_omitted>
   ```

2. The creation of token secrets by Kubernetes is no longer automatic in recent releases. It then falls on the user to design them as desired. Use the command shown below to create the token.

   ```
   student@cp:~$ export token=$(kubectl create token default)
   ```

3. 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@cp:~$ curl https://k8scp: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>
```

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

```
student@cp:~$ curl https://k8scp:6443/api/v1 --header "Authorization: Bearer $token" -k
```

```
<output_omitted>
```

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

```
student@cp:~$ curl \
     https://k8scp:6443/api/v1/namespaces --header "Authorization: Bearer $token" -k
```

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

6. 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@cp:~$ kubectl run -i -t busybox --image=busybox --restart=Never
```

Inside container

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

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

```
student@cp:~$ 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.

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

```
student@cp:~$ 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>
```

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

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

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

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

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

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

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

```
student@cp:~$ kill 22500
```

Exercise 6.3: Working with Jobs

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

Create A Job

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

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

job.yaml

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: sleepy
spec:
  template:
    spec:
      containers:
      - name: resting
        image: busybox
        command: ["/bin/sleep"]
        args: ["3"]
      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@cp:~$ kubectl create -f job.yaml
```

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

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

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

```
student@cp:~$ 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:     Thu, 23 Aug 2023 10:47:53 +0000
Completed At:   Thu, 23 Aug 2023 10:48:00 +0000
Duration:       5s
Pods Statuses:  0 Running / 1 Succeeded / 0 Failed
<output_omitted>
```

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

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

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

```
student@cp:~$ 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>
```

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

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

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

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

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

job.yaml

```yaml
<output_omitted>
metadata:
  name: sleepy
spec:
  completions: 5    #<--Add this line
  template:
    spec:
      containers:
<output_omitted>
```

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

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

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

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

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

```
student@cp:~$ 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@cp:˜$ vim job.yaml
```

job.yaml

```yaml
<output_omitted>
  name: sleepy
spec:
  completions: 5
  parallelism: 2   #<-- Add this line
  template:
    spec:
<output_omitted>
```

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

```
student@cp:˜$ kubectl create -f job.yaml
```

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

```
student@cp:˜$ 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@cp:˜$ 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@cp:˜$ vim job.yaml
```

```yaml
<output_omitted>
  completions: 5
  parallelism: 2
  activeDeadlineSeconds: 15   #<-- Add this line
  template:
    spec:
      containers:
      - name: resting
        image: busybox
        command: ["/bin/sleep"]
        args: ["5"]           #<-- Edit this line
<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@cp:˜$ kubectl delete jobs.batch sleepy
```

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

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

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

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

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

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

```
<output_omitted>
status:
  conditions:
  - lastProbeTime: 2023-08-23T10:48:00Z
    lastTransitionTime: 2023-08-23T10:48:00Z
    message: Job was active longer than specified deadline
    reason: DeadlineExceeded
    status: "True"
    type: Failed
  failed: 2
  startTime: 2023-08-23T10:48:00Z
  succeeded: 3
```

14. Delete the job.

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

```yaml
apiVersion: batch/v1
kind: CronJob                #<-- Update this line to CronJob
metadata:
```

```yaml
  name: sleepy
spec:
  schedule: "*/2 * * * *"     #<-- Add Linux style cronjob syntax
  jobTemplate:                #<-- New jobTemplate and spec move
    spec:
      template:               #<-- This and following lines move
        spec:                 #<-- four spaces to the right
          containers:
          - name: resting
            image: busybox
            command: ["/bin/sleep"]
            args: ["5"]
          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@cp:~$ kubectl create -f cronjob.yaml
```

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

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

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

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

```
No resources found.
```

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

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

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

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

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

```
student@cp:~$ 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
```

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

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

```yaml
....
  jobTemplate:
    spec:
```



```yaml
      template:
        spec:
          activeDeadlineSeconds: 10  #<-- Add this line
          containers:
          - name: resting
....
        command: ["/bin/sleep"]
        args: ["30"]                 #<-- Edit this line
      restartPolicy: Never
....
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

7. Clean up by deleting the `CronJob`.

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

rs.yaml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: rs-one
spec:
  replicas: 2
  selector:
    matchLabels:
      system: ReplicaOne
  template:
    metadata:
      labels:
        system: ReplicaOne
    spec:
      containers:
      - name: nginx
        image: nginx:1.15.1
        ports:
        - containerPort: 80
```

3. Create the `ReplicaSet`:

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

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

4. View the newly created `ReplicaSet`:

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

```
Name:           rs-one
Namespace:      default
Selector:       system=ReplicaOne
Labels:         <none>
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:1.15.1
    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@cp:~$ 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@cp:~$ kubectl delete rs rs-one --cascade=orphan
```

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

7. View the ReplicaSet and Pods again:

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

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

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

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

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

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

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

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

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

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

```
student@cp:~$ 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@cp:~$ 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@cp:~$ kubectl delete rs rs-one
```

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

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

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

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

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

15. Delete the remaining Pod using the label.

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

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

Exercise 7.2: Working with Deployments

A Deployment is a watch loop object which we have been working with in the previous labs. A Deployment provides a declarative update to Pods and ReplicaSets and ensure a particular number of pods are created in general, several could be on a single node. Deployment is a high-level resource object that is used to manage the rollout and scaling of containerized applications. A Deployment describes the desired state of the application, such as the number of replicas, and the container image to use. When a Deployment is created, Kubernetes will automatically create and manage the necessary replica sets, which in turn will create and manage the necessary pods to ensure that the desired state of the application is met. Deployment also provide rolling updates, which allow for updating an application to a new version without downtime by gradually replacing the old replicas with new ones. Using Deployment in Kubernetes makes it easy to manage and scale containerized applications while ensuring high availability and reliability.

1. We begin by creating a yaml file. In this case the kind would be set to deployment. We can generate the yaml file using the imperative method

```
student@cp:~$ kubectl create deploy webserver --image nginx:1.22.1 --replicas=2 \
               --dry-run=client -o yaml | tee dep.yaml
```

```
student@cp:~$ cat dep.yaml
```


dep.yaml

```yaml
....
kind: Deployment
....
name: webserver
....
replicas: 2
....
     app: webserver
....
```

2. Create and verify the newly formed Deployment. There should be two replicas of Pods created in the cluster.

```
student@cp:~$ kubectl create -f dep.yaml
```

```
deployment.apps/webserver created
```

```
student@cp:~$ kubectl get deploy
```

```
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
webserver   2/2     2            2           14s
```

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

```
NAME                          READY   STATUS    RESTARTS   AGE
webserver-6cbc654ddc-lssbm    1/1     Running   0          42s
webserver-6cbc654ddc-xpmtl    1/1     Running   0          42s
```

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

```
student@cp:~$ kubectl describe pod webserver-6cbc654ddc-lssbm  | grep Image:
```

```
Image:              nginx:1.22.1
```

Exercise 7.3: Rolling Updates and Rollbacks using Deployment

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 `recreate` 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 verified on the repositories on the registry

1. Begin by viewing the current `strategy` setting for the `Deployment` created in the previous section.

```
student@cp:~$ kubectl get deploy webserver -o yaml | grep -A 4 strategy
```

```
    strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
```

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

```
student@cp:~$ kubectl edit deploy webserver
```

```
....
strategy:
rollingUpdate:              # <-- remove this line
  maxSurge: 25%             # <-- remove this line
  maxUnavailable: 25%       # <-- remove this line
type: Recreate         # <-- Edit this line
:q....
```

3. Update the `Deployment` to use a newer version of the **nginx** server. This time use the **set** command instead of **edit**. Set the version to be `1.23.1-alpine`.

```
student@cp:~$ kubectl set image deploy webserver nginx=nginx:1.23.1-alpine
```

```
deployment.apps/webserver image updated
```

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

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

```
      NAME                             READY   STATUS    RESTARTS   AGE
webserver-6cf9cd5c74-qjph4   1/1     Running   0          35s
webserver-6cf9cd5c74-zc6x9   1/1     Running   0          35s
```

```
student@cp:~$ kubectl describe po webserver-6cf9cd5c74-qjph4 |grep Image:
```

```
Image:          nginx:1.23.1-alpine
```

5. View the history of changes for the `Deployment`. You should see two revisions listed. As we did not add the the `change-cause annotation` we didn't see why the object updated.

```
student@cp:~$ kubectl rollout history deploy webserver
```

```
deployment.apps/webserver
REVISION  CHANGE-CAUSE
1         <none>
2         <none>
```

6. View the settings for the various versions of the `Deployment`. The `Image:` line should be the only difference between the two outputs.

```
student@cp:~$ kubectl rollout history deploy webserver --revision=1
```

```
deployment.apps/webserver with revision #1
Pod Template:
  Labels:       app=webserver
        pod-template-hash=6cbc654ddc
  Containers:
   nginx:
    Image:      nginx:1.22.1
    Port:       <none>
    Host Port:  <none>
    Environment:        <none>
    Mounts:     <none>
  Volumes:      <none>
```

```
student@cp:~$ kubectl rollout history deploy webserver --revision=2
```

```
....
  Image:      nginx:1.23.1-alpine
.....
```

7. Use `kubectl rollout undo` to change the `Deployment` back to previous version.

```
student@cp:~$ kubectl rollout undo deploy webserver
```

```
deployment.apps/webserver rolled back
```

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

```
 NAME                          READY   STATUS    RESTARTS   AGE
webserver-6cbc654ddc-7wb5q    1/1     Running   0          37s
webserver-6cbc654ddc-svbtj    1/1     Running   0          37s
```

```
student@cp:~$ kubectl describe pod webserver-6cbc654ddc-7wb5q |grep Image:
```

```
Image:            nginx:1.22.1
```

8. Clean up the system by removing the `Deployment`.

```
student@cp:~$ kubectl delete deploy webserver
```

```
deployment.apps "webserver" deleted
```

Exercise 7.4: 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 pod 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@cp:~$ cp rs.yaml ds.yaml
```

```
student@cp:~$ vim ds.yaml
```

YAML **ds.yaml**

```yaml
....
kind: DaemonSet
....
  name: ds-one
....
  replicas: 2 #<<<----Remove this line
....
```

```yaml
        system: DaemonSetOne #<<-- Edit both references
....
```

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

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

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

```
student@cp:~$ kubectl get ds
```

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

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

```
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@cp:~$ kubectl describe pod ds-one-b1dcv | grep Image:
```

```
Image:                 nginx:1.15.1
```

Exercise 7.5: Rolling Updates and Rollbacks using DaemonSet

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@cp:~$ kubectl get ds ds-one -o yaml | grep -A 4 Strategy
```

```
  updateStrategy:
    rollingUpdate:
      maxSurge:; 0
      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@cp:~$ kubectl edit ds ds-one
```

```
....
  updateStrategy:
    rollingUpdate:
```

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

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

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

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

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

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

```
Image:                 nginx:1.15.1
```

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

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

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

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

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

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

```
Image:                 nginx:1.16.1-alpine
```

6. View the image running on the older Pod. It should still show version 1.15.1.

```
student@cp:~$ kubectl describe pod ds-one-z31r4 |grep Image:
```

```
Image:                 nginx:1.15.1
```

7. View the history of changes for the `DaemonSet`. You should see two revisions listed. As we did not add the the `change-cause annotation` we didn't see why the object updated.

```
student@cp:~$ 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@cp:~$ kubectl rollout history ds ds-one --revision=1
```

```
daemonsets "ds-one" with revision #1
Pod Template:
  Labels:       system=DaemonSetOne
  Containers:
```

```
      nginx:
       Image:        nginx:1.15.1
       Port:        80/TCP
       Environment:        <none>
       Mounts:        <none>
     Volumes:        <none>
```

```
student@cp:~$ kubectl rollout history ds ds-one --revision=2
```

```
....
    Image:        nginx:1.16.1-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@cp:~$ kubectl rollout undo ds ds-one --to-revision=1
```

```
daemonset.apps/ds-one rolled back
```

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

```
Image:                nginx:1.16.1-alpine
```

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

```
student@cp:~$ kubectl delete pod ds-one-xc86w
```

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

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

```
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@cp:~$ kubectl describe po ds-one-qc72k |grep Image:
```

```
Image:                nginx:1.15.1
```

11. View the details of the `DaemonSet`. The Image should be v1.15.1 in the output.

```
student@cp:~$ kubectl describe ds |grep Image:
```

```
Image:                nginx:1.15.1
```

12. View the current configuration for the `DaemonSet` in YAML output. Look for the `updateStrategy:` the the `type:`

```
student@cp:~$ kubectl get ds ds-one -o yaml
```

```
apiVersion: apps/v1
kind: DaemonSet
.....
      terminationGracePeriodSeconds: 30
  updateStrategy:
    type: OnDelete
```



```
status:
  currentNumberScheduled: 2
.....
```

13. Create a new `DaemonSet`, this time setting the update policy to `RollingUpdate`. Begin by generating a new config file.

```
student@cp:~$ kubectl get ds ds-one -o yaml  > ds2.yaml
```

14. Edit the file. Change the name, around line 69 and the update strategy around line 100, back to the default `RollingUpdate`.

```
student@cp:~$ vim ds2.yaml
```

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

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

```
student@cp:~$ kubectl create -f ds2.yaml
```

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

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

```
NAME                   READY     STATUS    RESTARTS   AGE
ds-one-qc72k           1/1       Running   0          28m
ds-one-z31r4           1/1       Running   0          57m
ds-two-10khc           1/1       Running   0          5m
ds-two-kzp9g           1/1       Running   0          5m
```

```
student@cp:~$ kubectl describe po ds-two-10khc |grep Image:
```

```
Image:                nginx:1.15.1
```

16. Edit the configuration file and set the image to a newer version such as 1.16.1-alpine.

```
student@cp:~$ kubectl edit ds ds-two
```

```
....
      - image: nginx:1.16.1-alpine
.....
```

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

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

```
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-81x7k            1/1       Running   0          32s
```

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

```
student@cp:~$ kubectl describe po ds-two-81x7k |grep Image:
```

```
Image:                nginx:1.16.1-alpine
```

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

```
student@cp:~$ kubectl rollout status ds ds-two
```

```
daemon set "ds-two" successfully rolled out
```

```
student@cp:~$ 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@cp:~$ kubectl rollout history ds ds-two --revision=2
```

```
...
    Image:        nginx:1.16.1-alpine
```

22. Clean up the system by removing the DaemonSets.

```
student@cp:~$ kubectl delete ds ds-one ds-two
```

```
daemonset.apps "ds-one" deleted
daemonset.apps "ds-two" deleted
```

Chapter 8

Volumes and Data

8.1 Labs

Exercise 8.1: Create a ConfigMap

Overview

Container files are ephemeral, which can be problematic for some applications. Should a container be restarted the files will be lost. In addition, we need a method to share files between containers inside a Pod.

A `Volume` is a directory accessible to containers in a Pod. Cloud providers offer volumes which persist further than the life of the Pod, such that AWS or GCE volumes could be pre-populated and offered to Pods, or transferred from one Pod to another. **Ceph** is also another popular solution for dynamic, persistent volumes.

Unlike current **Docker** volumes a Kubernetes volume has the lifetime of the Pod, not the containers within. You can also use different types of volumes in the same Pod simultaneously, but Volumes cannot mount in a nested fashion. Each must have their own mount point. Volumes are declared with `spec.volumes` and mount points with `spec.containers.volumeMounts` parameters. Each particular volume type, 24 currently, may have other restrictions. https://kubernetes.io/docs/concepts/storage/volumes/#types-of-volumes

We will also work with a `ConfigMap`, which is basically a set of key-value pairs. This data can be made available so that a Pod can read the data as environment variables or configuration data. A `ConfigMap` is similar to a `Secret`, except they are not base64 byte encoded arrays. They are stored as strings and can be read in serialized form.

There are three different ways a `ConfigMap` can ingest data, from a literal value, from a file or from a directory of files.

1. We will create a `ConfigMap` containing primary colors. We will create a series of files to ingest into the `ConfigMap`. First, we create a directory `primary` and populate it with four files. Then we create a file in our home directory with our favorite color.

```
student@cp:~$ mkdir primary

student@cp:~$ echo c > primary/cyan

student@cp:~$ echo m > primary/magenta
```

```
student@cp:~$ echo y > primary/yellow

student@cp:~$ echo k > primary/black

student@cp:~$ echo "known as key" >> primary/black

student@cp:~$ 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@cp:~$ 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@cp:~$ kubectl get configmap colors
```

```
NAME      DATA      AGE
colors    6         30s
```

```
student@cp:~$ 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@cp:~$ vim simpleshell.yaml
```

simpleshell.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: shell-demo
spec:
  containers:
  - name: nginx
    image: nginx
    env:
```

```yaml
    - name: ilike
      valueFrom:
        configMapKeyRef:
          name: colors
          key: favorite
```

5. Create the Pod and view the environmental variable. After you view the parameter, exit out and delete the pod.

```
student@cp:~$ kubectl create -f simpleshell.yaml
```

```
pod/shell-demo created
```

```
student@cp:~$ kubectl exec shell-demo -- /bin/bash -c 'echo $ilike'
```

```
blue
```

```
student@cp:~$ 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@cp:~$ vim simpleshell.yaml
```

simpleshell.yaml

```yaml
<output_omitted>
    image: nginx
#    env:
#    - name: ilike
#      valueFrom:
#        configMapKeyRef:
#          name: colors
#          key: favorite
    envFrom:                    #<-- Same indent as image: line
    - configMapRef:
        name: colors
```

```
student@cp:~$ kubectl create -f simpleshell.yaml
```

```
pod/shell-demo created
```

```
student@cp:~$ kubectl exec shell-demo -- /bin/bash -c 'env'
```

```
black=k
known as key

KUBERNETES_SERVICE_PORT_HTTPS=443
cyan=c
<output_omitted>
```

```
student@cp:~$ 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@cp:˜$ vim car-map.yaml
```

car-map.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: fast-car
  namespace: default
data:
  car.make: Ford
  car.model: Mustang
  car.trim: Shelby
```

8. Create the `ConfigMap` and verify the settings.

```
student@cp:˜$ kubectl create -f car-map.yaml
```

```
configmap/fast-car created
```

```
student@cp:˜$ kubectl get configmap fast-car -o yaml
```

```yaml
apiVersion: v1
data:
  car.make: Ford
  car.model: Mustang
  car.trim: Shelby
kind: ConfigMap
<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@cp:˜$ vim simpleshell.yaml
```

simpleshell.yaml

```yaml
<output_omitted>
spec:
  containers:
    - name: nginx
      image: nginx
      volumeMounts:
      - name: car-vol
        mountPath: /etc/cars
  volumes:
    - name: car-vol
      configMap:
        name: fast-car
<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@cp:~$ kubectl create -f simpleshell.yaml
```

```
pod "shell-demo" created
```

```
student@cp:~$  kubectl exec shell-demo -- /bin/bash -c 'df -ha |grep car'
```

```
/dev/root       9.6G  3.2G   6.4G  34% /etc/cars
```

```
student@cp:~$ 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@cp:~$ kubectl delete pods shell-demo
```

```
pod "shell-demo" deleted
```

```
student@cp:~$ kubectl delete configmap fast-car colors
```

```
configmap "fast-car" deleted
configmap "colors" deleted
```

Exercise 8.2: Creating a Persistent NFS Volume (PV)

We will first deploy an NFS server. Once tested we will create a persistent NFS volume for containers to claim.

1. Install the software on your cp node.

```
student@cp:~$ 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@cp:~$ sudo mkdir /opt/sfw
```

```
student@cp:~$ sudo chmod 1777 /opt/sfw/
```

```
student@cp:~$ 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@cp:~$ sudo vim /etc/exports
```

```
/opt/sfw/ *(rw,sync,no_root_squash,subtree_check)
```

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

```
student@cp:~$ sudo exportfs -ra
```

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

```
student@worker:~$ sudo apt-get -y install nfs-common
```

```
<output_omitted>
```

```
student@worker:~$ showmount -e k8scp
```

```
Export list for k8scp:
/opt/sfw  *
```

```
student@worker:~$ sudo mount k8scp:/opt/sfw /mnt
```

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

```
total 4
-rw-r--r-- 1 root root 23 Aug 28 17:55 hello.txt
```

6. Return to the cp node and create a YAML file for the object with `kind`, `PersistentVolume`. Use the hostname of the cp 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@cp:~$ vim PVol.yaml
```

PVol.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pvvol-1
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  nfs:
    path: /opt/sfw
    server: cp   #<-- Edit to match cp node
    readOnly: false
```

7. Create the persistent volume, then verify its creation.

```
student@cp:~$ kubectl create -f PVol.yaml
```

```
persistentvolume/pvvol-1 created
```

```
student@cp:~$ kubectl get pv
```

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

Exercise 8.3: Creating a Persistent Volume Claim (PVC)

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

1. Begin by determining if any currently exist.

```
student@cp:~$ kubectl get pvc
```

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

2. Create a YAML file for the new pvc.

```
student@cp:~$ vim pvc.yaml
```

pvc.yaml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-one
spec:
  accessModes:
  - ReadWriteMany
  resources:
     requests:
       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@cp:~$ kubectl create -f pvc.yaml
```

```
persistentvolumeclaim/pvc-one created
```

```
student@cp:~$ 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@cp:~$ kubectl get pv
```

```
NAME      CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS    CLAIM
   STORAGECLASS   REASON   AGE
pvvol-1   1Gi        RWX           Retain          Bound     default/pvc-one
                           5m
```

5. Create a new deployment to use the pvc. We will copy and edit an existing deployment yaml file. We will change the deployment name then add a volumeMounts section under containers and a 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@cp:~$ cp first.yaml nfs-pod.yaml
```

```
student@cp:~$ vim nfs-pod.yaml
```

nfs-pod.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
```

```yaml
  generation: 1
  labels:
    run: nginx
  name: nginx-nfs                  #<-- Edit name
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      run: nginx
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        run: nginx
    spec:
      containers:
      - image: nginx
        imagePullPolicy: Always
        name: nginx
        volumeMounts:              #<--  Add these three lines
        - name: nfs-vol
          mountPath: /opt
        ports:
        - containerPort: 80
          protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      volumes:                        #<-- Add these four lines
      - name: nfs-vol
        persistentVolumeClaim:
          claimName: pvc-one
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
```

6. Create the pod using the newly edited file.

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

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

```
student@cp:~$ kubectl describe pod nginx-nfs-1054709768-s8g28
```

```
Name:           nginx-nfs-1054709768-s8g28
Namespace:      default
Priority:       0
Node:           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@cp:~$ kubectl get pvc
```

```
NAME     STATUS VOLUME  CAPACITY ACCESS MODES  STORAGECLASS  AGE
pvc-one  Bound  pvvol-1 1Gi      RWX                         2m
```

Exercise 8.4: Using a ResourceQuota to Limit PVC Count and Usage

The flexibility of cloud-based storage often requires limiting consumption among users. We will use the `ResourceQuota` object to both limit the total consumption as well as the number of persistent volume claims.

1. Begin by deleting the deployment we had created to use NFS, the pv and the pvc.

```
student@cp:~$ kubectl delete deploy nginx-nfs
```

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

```
student@cp:~$ kubectl delete pvc pvc-one
```

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

```
student@cp:~$ 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@cp:~$ vim storage-quota.yaml
```

YAML storage-quota.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: storagequota
spec:
  hard:
```

```yaml
    persistentvolumeclaims: "10"
    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@cp:~$ kubectl create namespace small
```

```
namespace/small created
```

```
student@cp:~$ 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@cp:~$ kubectl -n small create -f PVol.yaml
```

```
persistentvolume/pvvol-1 created
```

```
student@cp:~$ 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@cp:~$ 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@cp:~$ 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@cp:˜$ vim nfs-pod.yaml
```

8. Create the container again.

```
student@cp:˜$ kubectl -n small create -f nfs-pod.yaml
```

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

9. Determine if the deployment has a running pod.

```
student@cp:˜$ kubectl -n small get deploy
```

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

```
student@cp:˜$ kubectl -n small describe deploy nginx-nfs
```

```
<output_omitted>
```

10. Look to see if the pods are ready.

```
student@cp:˜$ 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@cp:˜$ 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@cp:˜$ 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@cp:˜$ 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@cp:˜$ kubectl describe ns small
```

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

```
student@cp:˜$ 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@cp:˜$ kubectl -n small get deploy
```

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

```
student@cp:˜$ 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@cp:˜$ 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@cp:˜$ kubectl -n small get pvc
```

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

```
student@cp:˜$ kubectl -n small delete pvc pvc-one
```

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

```
student@cp:~$ 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@cp:~$ kubectl get pv/pvvol-1 -o yaml
```

```yaml
....
    path: /opt/sfw
    server: cp
  persistentVolumeReclaimPolicy: Retain
status:
  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@cp:~$ kubectl delete pv/pvvol-1
```

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

```
student@cp:~$ grep Retain PVol.yaml
```

```
persistentVolumeReclaimPolicy: Retain
```

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

```
persistentvolume/pvvol-1 patched
```

```
student@cp:~$ 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@cp:~$ kubectl describe ns small
```

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

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

```
student@cp:~$ kubectl -n small create -f pvc.yaml
```

```
persistentvolumeclaim/pvc-one created
```

```
student@cp:~$ kubectl describe ns small
```

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

22. Remove the existing quota from the namespace.

```
student@cp:~$ kubectl -n small get resourcequota
```

```
NAME           CREATED AT
storagequota   2023-08-25T04:10:02Z
```

```
student@cp:~$ kubectl -n small delete resourcequota storagequota
```

```
resourcequota "storagequota" deleted
```

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

```
student@cp:~$ vim storage-quota.yaml
```

YAML

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

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

```
student@cp:~$ kubectl -n small create -f storage-quota.yaml
```

```
resourcequota/storagequota created
```

```
student@cp:~$ 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@cp:~$ kubectl -n small create -f nfs-pod.yaml
```

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

```
student@cp:~$ 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@cp:~$ 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@cp:~$ kubectl -n small delete deploy nginx-nfs
```

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

```
student@cp:~$ 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@cp:~$ 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@cp:~$ kubectl delete pv/pvvol-1
```

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

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

```
student@cp:~$ vim PVol.yaml
```

YAML PVol.yaml

```yaml
....
  persistentVolumeReclaimPolicy: Recycle
....
```

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@cp:~$ 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@cp:~$ 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@cp:~$ kubectl -n small create -f PVol.yaml
```

```
persistentvolume/pvvol-1 created
```

```
student@cp:~$ 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@cp:~$ 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@cp:~$ kubectl -n small edit resourcequota
```

```yaml
....
spec:
  hard:
    persistentvolumeclaims: "10"
    requests.storage: 500Mi
status:
  hard:
    persistentvolumeclaims: "10"
....
```

36. Create the `pvc` again. It should work this time. Then create the `deployment` again.

```
student@cp:~$ kubectl -n small create -f pvc.yaml
```

```
persistentvolumeclaim/pvc-one created
```

```
student@cp:~$ kubectl -n small create -f nfs-pod.yaml
```

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

37. View the namespace settings.

```
student@cp:~$ kubectl describe ns small
```

```
<output_omitted>
```

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

```
student@cp:~$ kubectl -n small delete deploy nginx-nfs
```

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

```
student@cp:~$ kubectl -n small get pvc
```

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

```
student@cp:~$ 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@cp:~$ kubectl -n small delete pvc pvc-one
```

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

```
student@cp:~$ 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@cp:~$ kubectl delete pv pvvol-1
```

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

Chapter 9

Services

9.1 Labs

Exercise 9.1: Deploy A New Service

Overview

Services (also called **microservices**) are objects which declare a policy to access a logical set of Pods. They are typically assigned with `labels` to allow persistent access to a resource, when front or back end containers are terminated and replaced.

Native applications can use the `Endpoints` API for access. Non-native applications can use a Virtual IP-based bridge to access back end pods. `ServiceTypes Type` could be:

- **ClusterIP** default - exposes on a cluster-internal IP. Only reachable within cluster
- **NodePort** Exposes node IP at a static port. A ClusterIP is also automatically created.
- **LoadBalancer** Exposes service externally using cloud providers load balancer. `NodePort` and `ClusterIP` automatically created.
- **ExternalName** Maps service to contents of `externalName` using a `CNAME` record.

We use services as part of decoupling such that any agent or object can be replaced without interruption to access from client to back end application.

1. Deploy two **nginx** servers using **kubectl** and a new `.yaml` file. The kind should be `Deployment` and label it with `nginx`. Create two replicas and expose port 8080. What follows is a well documented file. There is no need to include the comments when you create the file. This file can also be found among the other examples in the tarball.

```
student@cp:~$ vim nginx-one.yaml
```

nginx-one.yaml

```yaml
apiVersion: apps/v1
# Determines YAML versioned schema.
kind: Deployment
# Describes the resource defined in this file.
metadata:
  name: nginx-one
  labels:
    system: secondary
# Required string which defines object within namespace.
  namespace: accounting
# Existing namespace resource will be deployed into.
spec:
  selector:
    matchLabels:
      system: secondary
# Declaration of the label for the deployment to manage
  replicas: 2
# How many Pods of following containers to deploy
  template:
    metadata:
      labels:
        system: secondary
# Some string meaningful to users, not cluster. Keys
# must be unique for each object. Allows for mapping
# to customer needs.
    spec:
      containers:
# Array of objects describing containerized application with a Pod.
# Referenced with shorthand spec.template.spec.containers
        - image: nginx:1.20.1
# The Docker image to deploy
          imagePullPolicy: Always
          name: nginx
# Unique name for each container, use local or Docker repo image
          ports:
          - containerPort: 8080
            protocol: TCP
# Optional resources this container may need to function.
      nodeSelector:
        system: secondOne
# One method of node affinity.
```

2. View the existing labels on the nodes in the cluster.

```
student@cp:~$ 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 about the `accounting` namespace.

```
student@cp:~$ 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@cp:~$ kubectl create ns accounting
```

```
namespace/accounting" created
```

```
student@cp:~$ 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@cp:~$ 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@cp:~$ 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. Note the value is case sensitive. Verify the labels.

```
student@cp:~$ kubectl label node <worker_node_name> system=secondOne
```

```
node/worker labeled
```

```
student@cp:~$ kubectl get nodes --show-labels
```

```
NAME     STATUS   ROLES                 AGE   VERSION   LABELS
cp       Ready    control-plane         15h   v1.28.1   beta.kubernetes.io/arch=amd64,
beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=cp,
kubernetes.io/os=linux,node-role.kubernetes.io/control-plane=,node-role.kubernetes.io/master=,
node.kubernetes.io/exclude-from-external-load-balancers=
worker   Ready    <none>                 15h   v1.28.1   beta.kubernetes.io/arch=amd64,
beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=worker,
kubernetes.io/os=linux,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@cp:~$  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@cp:~$ 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@cp:~$ 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@cp:~$ 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@cp:~$ curl 192.168.1.72:8080
```

```
curl: (7) Failed to connect to 192.168.1.72 port 8080: Connection refused
```

```
student@cp:~$ 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@cp:~$ kubectl -n accounting delete deploy nginx-one
```

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

```
student@cp:~$ vim nginx-one.yaml
```

nginx-one.yaml

```yaml
....
        ports:
        - containerPort: 8080     #<-- Edit this line
          protocol: TCP
....
```

```
student@cp:~$ kubectl create -f nginx-one.yaml
```

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

Exercise 9.2: Configure a NodePort

In a previous exercise we deployed a `LoadBalancer` which deployed a `ClusterIP` and`NodePort` automatically. In this exercise we will deploy a `NodePort`. While you can access a container from within the cluster, one can use a `NodePort` to NAT traffic from outside the cluster. One reason to deploy a `NodePort` instead, is that a `LoadBalancer` is also a load balancer resource from cloud providers like GKE and AWS.

1. In a previous step we were able to view the **nginx** page using the internal Pod IP address. Now expose the deployment using the `--type=NodePort`. We will also give it an easy to remember name and place it in the `accounting` namespace. We could pass the port as well, which could help with opening ports in the firewall.

```
student@cp:~$ 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@cp:~$ kubectl -n accounting describe services
```

```
....
NodePort:                 <unset>  32103/TCP
....
```

3. Locate the exterior facing hostname or IP address of the cluster. The lab assumes use of GCP nodes, which we access via a `FloatingIP`, we will first check the internal only public IP address. Look for the Kubernetes cp URL. Whichever way you access check access using both the internal and possible external IP address

```
student@cp:~$ kubectl cluster-info
```

```
Kubernetes control plane is running at https://k8scp:6443
CoreDNS is running at https://k8scp:6443/api/v1/namespaces/kube-system/services/kube-dns: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 cp URL and `NodePort`.

```
student@cp:~$ curl http://k8scp: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. You may be able to use **curl** to locate your public IP address.

```
student@cp:~$ curl ifconfig.io
```

```
104.198.192.84
```

Exercise 9.3: Working with CoreDNS

1. We can leverage **CoreDNS** and predictable hostnames instead of IP addresses. A few steps back we created the `service-lab` NodePort in the Accounting namespace. We will create a new pod for testing using `Ubuntu`. The pod name will be named `ubuntu`.

```
student@cp:~$ vim nettool.yaml
```

nettool.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: ubuntu
spec:
  containers:
  - name: ubuntu
    image: ubuntu:latest
    command: [ "sleep" ]
    args: [ "infinity" ]
```

2. Create the pod and then log into it.

```
student@cp:˜$ kubectl create -f nettool.yaml
```

```
pod/ubuntu created
```

```
student@cp:˜$ kubectl exec -it ubuntu -- /bin/bash
```

On Container

(a) Add some tools for investigating DNS and the network. The installation will ask you the geographic area and timezone information. Someone in Austin would first answer 2. America, then 37 for Chicago, which would be central time

```
root@ubuntu:/#  apt-get update ; apt-get install curl dnsutils -y
```

(b) Use the **dig** command with no options. You should see root name servers, and then information about the DNS server responding, such as the IP address.

```
root@ubuntu:/# dig
```

```
; <<>> DiG 9.16.1-Ubuntu <<>>
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 3394
;; flags: qr rd ra; QUERY: 1, ANSWER: 13, AUTHORITY: 0, ADDITIONAL: 1

<output_omitted>

;; Query time: 4 msec
;; SERVER: 10.96.0.10#53(10.96.0.10)
;; WHEN: Thu Aug 27 22:06:18 CDT 2020
;; MSG SIZE  rcvd: 431
```

(c) Also take a look at the /etc/resolv.conf file, which will indicate nameservers and default domains to search if no using a Fully Qualified Distinguished Name (FQDN). From the output we can see the first entry is `default.svc.cluster.local.`.

```
root@ubuntu:/# cat /etc/resolv.conf
```

```
nameserver 10.96.0.10
search default.svc.cluster.local svc.cluster.local cluster.local
c.endless-station-188822.internal google.internal
options ndots:5
```

(d) Use the **dig** command to view more information about the DNS server. Us the **-x** argument to get the FQDN using the IP we know. Notice the domain name, which uses `.kube-system.svc.cluster.local.`, to match the pod namespaces instead of `default`. Also note the name, `kube-dns`, is the name of a service not a pod.

```
root@ubuntu:/# dig @10.96.0.10 -x 10.96.0.10
```

```
...
;; QUESTION SECTION:
;10.0.96.10.in-addr.arpa.        IN        PTR

;; ANSWER SECTION:
10.0.96.10.in-addr.arpa.
→  30        IN        PTR        kube-dns.kube-system.svc.cluster.local.

;; Query time: 0 msec
;; SERVER: 10.96.0.10#53(10.96.0.10)
;; WHEN: Thu Aug 27 23:39:14 CDT 2020
;; MSG SIZE  rcvd: 139
```

(e) Recall the name of the `service-lab` service we made and the namespaces it was created in. Use this information to create a FQDN and view the exposed pod.

```
root@ubuntu:/# curl service-lab.accounting.svc.cluster.local.
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
...
```

(f) Attempt to view the default page using just the service name. It should fail as `nettool` is in the default namespace.

```
root@ubuntu:/# curl service-lab
```

```
curl: (6) Could not resolve host: service-lab
```

(g) Add the `accounting` namespaces to the name and try again. Traffic can access a service using a name, even across different namespaces.

```
root@ubuntu:/# curl service-lab.accounting
```

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

(h) Exit out of the container and look at the services running inside of the `kube-system` namespace. From the output we see that the `kube-dns` service has the DNS serverIP, and exposed ports DNS uses.

```
root@ubuntu:/# exit
```

```
student@cp:~$ kubectl -n kube-system get svc
```

```
NAME       TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)                  AGE
kube-dns   ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP   42h
```

3. Examine the service in detail. Among other information notice the selector in use to determine the pods the service communicates with.

```
student@cp:~$ kubectl -n kube-system get svc kube-dns -o yaml
```

```
...
    labels:
      k8s-app: kube-dns
      kubernetes.io/cluster-service: "true"
      kubernetes.io/name: CoreDNS
...
    selector:
      k8s-app: kube-dns
    sessionAffinity: None
    type: ClusterIP
...
```

4. Find pods with the same labels in all namespaces. We see that infrastructure pods all have this label, including **coredns**.

```
student@cp:~$ kubectl get pod -l k8s-app --all-namespaces
```

```
NAMESPACE     NAME                        READY   STATUS    RESTARTS   AGE
kube-system   cilium-5tv9d                1/1     Running   0          136m
kube-system   cilium-gzdk6                1/1     Running   0          54m
kube-system   coredns-5d78c9869d-44qvq    1/1     Running   0          31m
kube-system   coredns-5d78c9869d-j6tqx    1/1     Running   0          31m
kube-system   kube-proxy-lpsmq            1/1     Running   0          35m
kube-system   kube-proxy-pvl8w            1/1     Running   0          34m
```

5. Look at the details of one of the `coredns` pods. Read through the pod spec and find the image in use as well as any configuration information. You should find that configuration comes from a configmap.

```
student@cp:~$ kubectl -n kube-system get pod coredns-f9fd979d6-4dxpl -o yaml
```

```
...
spec:
  containers:
  - args:
    - -conf
    - /etc/coredns/Corefile
    image: k8s.gcr.io/coredns:1.7.0
...
    volumeMounts:
    - mountPath: /etc/coredns
      name: config-volume
      readOnly: true
...
  volumes:
  - configMap:
      defaultMode: 420
      items:
      - key: Corefile
        path: Corefile
      name: coredns
    name: config-volume
...
```


6. View the configmaps in the `kube-system` namespace.

```
student@cp:~$ kubectl -n kube-system get configmaps
```

```
NAME                                 DATA   AGE
cilium-config                        4      43h
coredns                              1      43h
extension-apiserver-authentication   6      43h
kube-proxy                           2      43h
kubeadm-config                       2      43h
kubelet-config                       1      43h
```

7. View the details of the `coredns` configmap. Note the `cluster.local` domain is listed.

```
student@cp:~$ kubectl -n kube-system get configmaps coredns -o yaml
```

```
apiVersion: v1
data:
  Corefile: |
    .:53 {
        errors
        health {
           lameduck 5s
        }
        ready
        kubernetes cluster.local in-addr.arpa ip6.arpa {
           pods insecure
           fallthrough in-addr.arpa ip6.arpa
           ttl 30
        }
        prometheus :9153
        forward . /etc/resolv.conf {
           max_concurrent 1000
        }
        cache 30
        loop
        reload
        loadbalance
    }
kind: ConfigMap
...
```

8. While there are many options and zone files we could configure, lets start with simple edit. Add a rewrite statement such that `test.io` will redirect to `cluster.local` More about each line can be found at coredns.io.

```
student@cp:~$ kubectl -n kube-system edit configmaps coredns
```

```
apiVersion: v1
data:
  Corefile: |
    .:53 {
        rewrite name regex (.*)\.test\.io {1}.default.svc.cluster.local   #<-- Add this line
        errors
        health {
           lameduck 5s
        }
        ready
        kubernetes cluster.local in-addr.arpa ip6.arpa {
           pods insecure
           fallthrough in-addr.arpa ip6.arpa
           ttl 30
        }
        prometheus :9153
```

```
        forward . /etc/resolv.conf {
           max_concurrent 1000
        }
        cache 30
        loop
        reload
        loadbalance
    }
```

9. Delete the coredns pods causing them to re-read the updated configmap.

```
student@cp:˜$ kubectl -n kube-system delete pod coredns-f9fd979d6-s4j98 coredns-f9fd979d6-xlpzf
```

```
pod "coredns-f9fd979d6-s4j98" deleted
pod "coredns-f9fd979d6-xlpzf" deleted
```

10. Create a new web server and create a `ClusterIP` service to verify the address works. Note the new service IP to start with a reverse lookup.

```
student@cp:˜$ kubectl create deployment nginx --image=nginx
```

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

```
student@cp:˜$ kubectl expose deployment nginx --type=ClusterIP --port=80
```

```
service/nginx expose
```

```
student@cp:˜$ kubectl get svc
```

```
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP   3d15h
nginx        ClusterIP   10.104.248.141   <none>        80/TCP    7s
```

11. Log into the `ubuntu` container and test the URL rewrite starting with the reverse IP resolution.

```
student@cp:˜$ kubectl exec -it ubuntu -- /bin/bash
```

On Container

(a) Use the **dig** command. Note that the service name becomes part of the FQDN.

```
root@ubuntu:/# dig -x 10.104.248.141
```

```
....
;; QUESTION SECTION:
;141.248.104.10.in-addr.arpa.        IN       PTR

;; ANSWER SECTION:
141.248.104.10.in-addr.arpa.
↪  30        IN        PTR       nginx.default.svc.cluster.local.
....
```

(b) Now that we have the reverse lookup test the forward lookup. The IP should match the one we used in the previous step.

```
root@ubuntu:/# dig nginx.default.svc.cluster.local.
```

```
....
;; QUESTION SECTION:
;nginx.default.svc.cluster.local. IN        A

;; ANSWER SECTION:
nginx.default.svc.cluster.local. 30 IN        A        10.104.248.141
....
```

(c) Now test to see if the rewrite rule for the `test.io` domain we added resolves the IP. Note the response uses the original name, not the requested FQDN.

```
root@ubuntu:/# dig nginx.test.io
```

```
....
;; QUESTION SECTION:
;nginx.test.io.                        IN        A

;; ANSWER SECTION:
nginx.default.svc.cluster.local. 30 IN        A        10.104.248.141
....
```

12. Exit out of the container then edit the configmap to add an `answer` section.

```
student@cp:~$ kubectl -n kube-system edit configmaps coredns
```

```
....
data:
  Corefile: |
    .:53 {
        rewrite stop {                        #<-- Edit this and following two lines
           name regex (.*)\.test\.io {1}.default.svc.cluster.local
           answer name (.*)\.default\.svc\.cluster\.local {1}.test.io
           }
        errors
        health {

....
```

13. Delete the coredns pods again to ensure they re-read the updated configmap.

```
student@cp:~$ kubectl -n kube-system delete pod coredns-f9fd979d6-fv9qn coredns-f9fd979d6-lnxn5
```

```
pod "coredns-f9fd979d6-fv9qn" deleted
pod "coredns-f9fd979d6-lnxn5" deleted
```

14. Log into the ubuntu container again. This time the response should show the FQDN with the requested FQDN.

```
student@cp:~$ kubectl exec -it ubuntu -- /bin/bash
```

On Container

```
root@ubuntu:/# dig nginx.test.io
```

```
....
;; QUESTION SECTION:
;nginx.test.io.                        IN        A

;; ANSWER SECTION:
nginx.test.io.                30        IN        A        10.104.248.141
....
```

15. Exit then delete the DNS test tools container to recover the resources.

```
student@cp:˜$ kubectl delete -f nettool.yaml
```

Exercise 9.4: Use Labels to Manage Resources

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

```
student@cp:˜$ kubectl delete pods -l system=secondary \
           --all-namespaces
```

```
pod "nginx-one-74dd9d578d-fcpmv" deleted
pod "nginx-one-74dd9d578d-sts5l" deleted
```

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

```
student@cp:˜$ 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@cp:˜$ 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@cp:˜$ 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@cp:˜$ kubectl label node worker system-
```

```
node/worker unlabeled
```

Chapter 10

Helm

10.1 Labs

Exercise 10.1: 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 `cp` node use **wget** to download the compressed tar file. Various versions can be found here: https://github.com/helm/helm/releases/

   ```
   student@cp:~$ wget https://get.helm.sh/helm-v3.12.0-linux-amd64.tar.gz
   ```

   ```
   <output_omitted>
   helm-v3.9.2-linux-a 100%[===================>]  13.35M  --.-KB/s    in 0.1s

   2023-06-01 03:18:50 (52.0 MB/s) - ‘helm-v3.12.0-linux-amd64.tar.gz’ saved [14168950/14168950]
   ```

2. Uncompress and expand the file.

   ```
   student@cp:~$ tar -xvf helm-v3.12.0-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@cp:˜$ sudo cp linux-amd64/helm /usr/local/bin/helm
```

4. A Chart is a collection of files to deploy an application. There is a good starting repo available on https://github.com/kubernetes/charts/tree/master/stable, provided by vendors, or you can make your own. Search the current Charts in the Helm Hub or an instance of Monocular for available stable databases. Repos change often, so the following output may be different from what you see.

```
student@cp:˜$ helm search hub database
```

```
URL                                                         CHART VERSION
                APP VERSION                   DESCRIPTION
https://artifacthub.io/packages/helm/drycc/data...          1.0.2
                                              A PostgreSQL database used by Drycc Workflow.
https://artifacthub.io/packages/helm/drycc-cana...          1.0.0
                                                    A PostgreSQL database used by Drycc
                                                    ↪  Workflow.
https://artifacthub.io/packages/helm/camptocamp...          0.0.6
                       1.0                             Expose services and secret to access postgres
                       ↪  d...
https://artifacthub.io/packages/helm/cnieg/h2-d...          1.0.3
                       1.4.199                         A helm chart to deploy h2-database
<output_omitted>
```

5. You can also add repositories from various vendors, often found by searching artifacthub.io such as ealenn, who has an echo program.

```
student@cp:˜$ helm repo add ealenn https://ealenn.github.io/charts
```

```
"ealenn" has been added to your repositories
```

```
student@cp:˜$ helm repo update
```

```
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "ealenn" chart repository
Update Complete.  Happy Helming!
```

6. We will install the **tester** tool. The **- -debug** option will create a lot of output. The output will typically suggest ways to access the software.

```
student@cp:˜$ helm upgrade -i tester ealenn/echo-server  --debug
```

```
history.go:56: [debug] getting history for release tester
Release "tester" does not exist. Installing it now.
install.go:173: [debug] Original chart version: ""
install.go:190: [debug] CHART PATH: /home/student/.cache/helm/repository/echo-server-0.5.0.tgz

client.go:122: [debug] creating 4 resource(s)
NAME: tester
<output_omitted>
```

7. Ensure the newly created tester-echo-server pod is running. Fix any issues, if not.

8. Look for the newly created service. Send a **curl** to the ClusterIP. You should get a lot of information returned.

```
student@cp:˜$ kubectl get svc
```

```
NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
kubernetes           ClusterIP   10.96.0.1       <none>        443/TCP    26h
tester-echo-server   ClusterIP   10.98.252.11    <none>        80/TCP     11m
```

```
student@cp:~$ curl 10.98.252.11
```


```
{"host":{"hostname":"10.98.252.11","ip":"::ffff:192.168.74.128","ips":
[]},"http":{"method":"GET","baseUrl":"","originalUrl":"/","protocol":
"http"},"request":{"params":{"0":"/"},"query":{},"cookies":{},"body":
{},"headers":{"host":"10.98.252.11","user-agent":"curl/7.58.0","accept":
"*/*"}},"environment":{"PATH":"/usr/local/sbin:/usr/local/bin:/usr/sbin:
/usr/bin:/sbin:/bin","TERM":"xterm","HOSTNAME":"tester-echo-server-
786768d9f4-4zsz9","ENABLE__HOST":"true","ENABLE__HTTP":"true","ENABLE__
<output_omitted>
```


9. View the `Chart` history on the system. The use of the **-a** option will show all `Charts` including `deleted` and `failed` attempts.

```
student@cp:~$ helm list
```

```
NAME            NAMESPACE       REVISION        UPDATED
STATUS          CHART                   APP VERSION
tester          default         1               2021-06-11 07:31:56.151628311 +0000 UTC
deployed        echo-server-0.5.0       0.6.0
```

10. Delete the **tester** `Chart`. No releases of tester should be found.

```
student@cp:~$ helm uninstall tester
```

```
release "tester" uninstalled
```

```
student@cp:~$ helm list
```

```
NAME    NAMESPACE   REVISION    UPDATED     STATUS      CHART       APP VERSION
```

11. Find the downloaded chart. It should be a compressed tarball under the user's home directory. Your **echo** version may be slightly different.

```
student@cp:~$ find $HOME -name *echo*
```

```
/home/student/.cache/helm/repository/echo-server-0.5.0.tgz
```

12. Move to the `archive` directory and extract the tarball. Take a look at the files within.

```
student@cp:~$ cd $HOME/.cache/helm/repository ; tar -xvf echo-server-*
```

```
echo-server/Chart.yaml
echo-server/values.yaml
echo-server/templates/_helpers.tpl
echo-server/templates/configmap.yaml
echo-server/templates/deployment.yaml
<output_omitted>
```

13. Examine the `values.yaml` file to see some of the values that could have been set.

```
student@cp:~/.cache/helm/repository$ cat echo-server/values.yaml
```

```
<output_omitted>
```

14. You can also download and examine or edit the values file before installation. Add another repo and download the Bitnami Apache chart.

```
student@cp:˜$ helm repo add bitnami https://charts.bitnami.com/bitnami

student@cp:˜$ helm fetch bitnami/apache --untar

student@cp:˜$ cd apache/
```

15. Take a look at the chart. You'll not it looks similar to the previous. Read through the :values.yaml:.

```
student@cp:˜$ ls
```

```
Chart.lock  Chart.yaml  README.md  charts  ci  files  templates
values.schema.json  values.yaml
```

```
student@cp:˜$ less values.yaml
```

```
## Global Docker image parameters
## Please, note that this will override the image parameters, including dependencies,
↪  configured....
## Current available global Docker image parameters: imageRegistry and imagepullSecrets
##
# global:
#   imageRegistry: myRegistryName
#   imagePullSecrets:
#     - myRegistryKeySecretName
<output_omitted>
```

16. Use the `values.yaml` file to install the chart. Take a look at the output and ensure the pod is running.

```
student@cp:˜$ helm install anotherweb .
```

```
NAME: anotherweb
LAST DEPLOYED: Fri Jun 11 08:11:10 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
<output_omitted>
```

17. Test the newly created service. You should get an HTML response saying `It works!` If the steps to find the service and check that it works are not familiar, you may want to make a note to review prior chapters.

18. Remove anything you have installed using **helm**. Reference earlier in the chapter if you don't remember the command. We will use **helm** again in another lab.

Chapter 11

Ingress

11.1 Labs

Exercise 11.1: Service Mesh

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. While nginx and GCE have controllers mentioned a lot in Kubernetes.io, there are many to chose from. Even more functionality and metrics come from the use of a service mesh, such as Istio, Linkerd, Contour, Aspen, or several others.

1. We will install linkerd using their own scripts. There is quite a bit of output. Instead of showing all of it the output has been omitted. Look through the output and ensure that everything gets a green check mark. Some steps may take a few minutes to complete. Each command is listed here to make install easier. As well these steps are in the `setupLinkerd.txt` file.

```
student@cp:~$ curl -sL run.linkerd.io/install | sh

student@cp:~$ export PATH=$PATH:/home/student/.linkerd2/bin

student@cp:~$ echo "export PATH=$PATH:/home/student/.linkerd2/bin" >> $HOME/.bashrc

student@cp:~$ linkerd check --pre

student@cp:~$ linkerd install --crds | kubectl apply -f -

student@cp:~$ linkerd install | kubectl apply -f -

student@cp:~$ linkerd check

student@cp:~$ linkerd viz install | kubectl apply -f -

student@cp:~$ linkerd viz check

student@cp:~$ linkerd viz dashboard &
```

2. By default the GUI is on available on the localhost. We will need to edit the service and the deployment to allow outside access, in case you are using a cloud provider for the nodes. Edit to remove all characters after equal sign for `-enforced-host`, which is around line 59.

```
student@cp:~$ kubectl -n linkerd-viz edit deploy web
```

```yaml
spec:
      containers:
      - args:
        - -linkerd-controller-api-addr=linkerd-controller-api.linkerd.svc.cluster.local:8085
        - -linkerd-metrics-api-addr=metrics-api.linkerd-viz.svc.cluster.local:8085
        - -cluster-domain=cluster.local
        - -grafana-addr=grafana.linkerd-viz.svc.cluster.local:3000
        - -controller-namespace=linkerd
        - -viz-namespace=linkerd-viz
        - -log-level=info
        - -enforced-host=                                 #<-- Remove everything after equal sign
        image: cr.l5d.io/linkerd/web:stable-2.11.1
        imagePullPolicy: IfNotPresent
```

3. Now edit the http nodePort and type to be a NodePort.

```
student@cp:~$ kubectl edit svc web -n linkerd-viz
```

```yaml
....
ports:
  - name: http
    nodePort: 31500                                  #<-- Add line with an easy to remember port
    port: 8084
....
  sessionAffinity: None
  type: NodePort                                     #<-- Edit type to be NodePort
status:
  loadBalancer: {}
....
```

4. Test access using a local browser to your public IP. Your IP will be different than the one shown below.

```
student@cp:~$ curl ifconfig.io
```

```
104.197.159.20
```

5. From you local system open a browser and go to the public IP and the high-number nodePort. Be aware the look of the web page may look slightly different as the software is regularly updated, for example Grafana is not longer fully integrated.

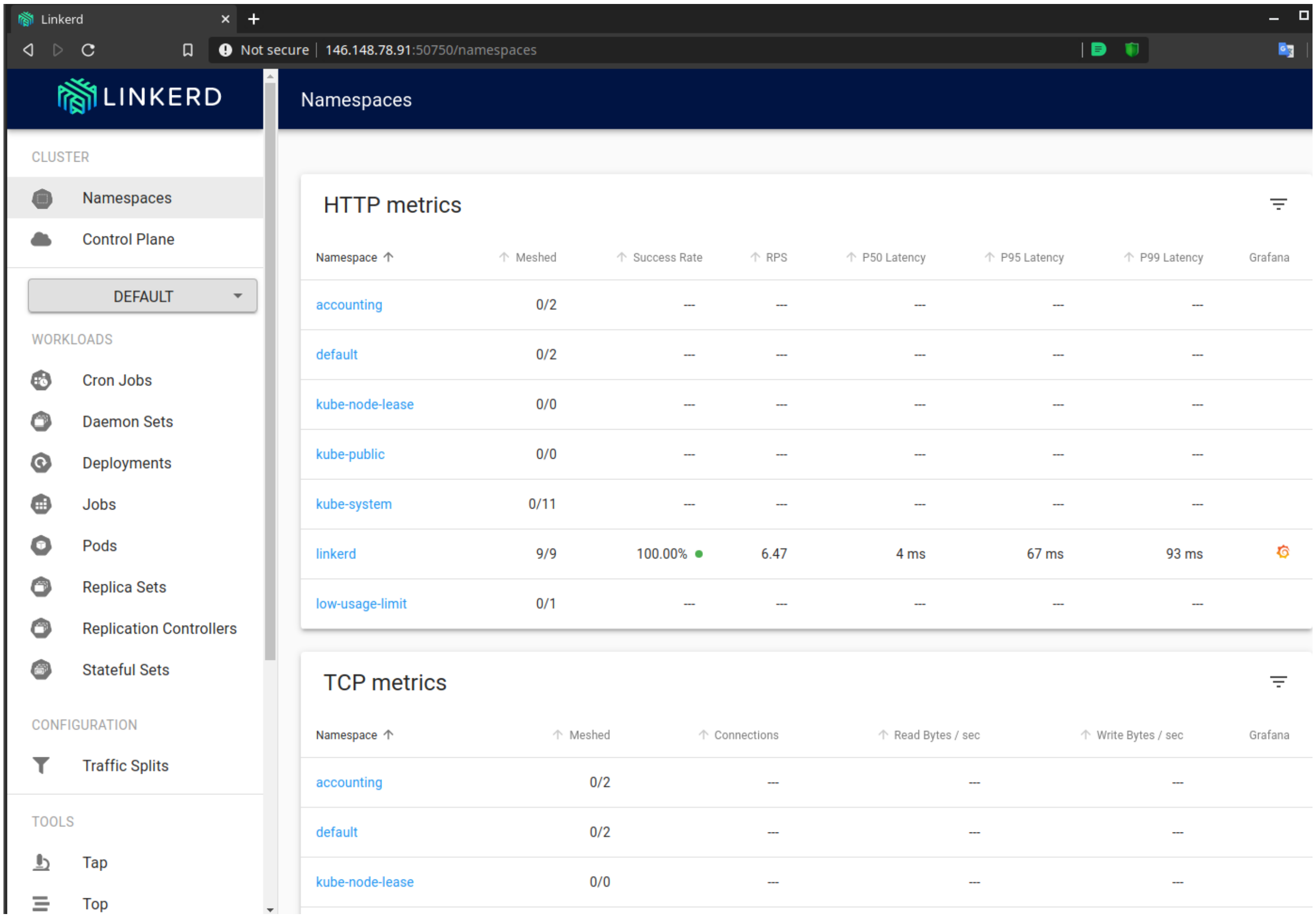


Figure 11.1: **Main Linkerd Page**

6. In order for linkerd to pay attention to an object we need to add an annotation. The **linkerd inject** command will do this for us. Generate YAML and pipe it to **linkerd** then pipe again to **kubectl**. Expect an error about how the object was created, but the process will work. The command can run on one line if you omit the back-slash. Recreate the `nginx-one` deployment we worked with in a previous lab exercise.

```
student@cp:~$  kubectl -n accounting get deploy nginx-one -o yaml | \
     linkerd inject - | kubectl apply -f -
```

```
<output_omitted>
```

7. Check the GUI, you should see that the `accounting` namespaces and pods are now meshed, and the name is a link.

8. Generate some traffic to the pods, and watch the traffic via the GUI. Use the `service-lab` service.

```
student@cp:~$ kubectl -n accounting get svc
```

```
NAME          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
nginx-one     ClusterIP   10.107.141.227   <none>        8080/TCP       5h15m
service-lab   NodePort    10.102.8.205     <none>        80:30759/TCP   5h14m
```

```
student@cp:~$ curl 10.102.8.205
```

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

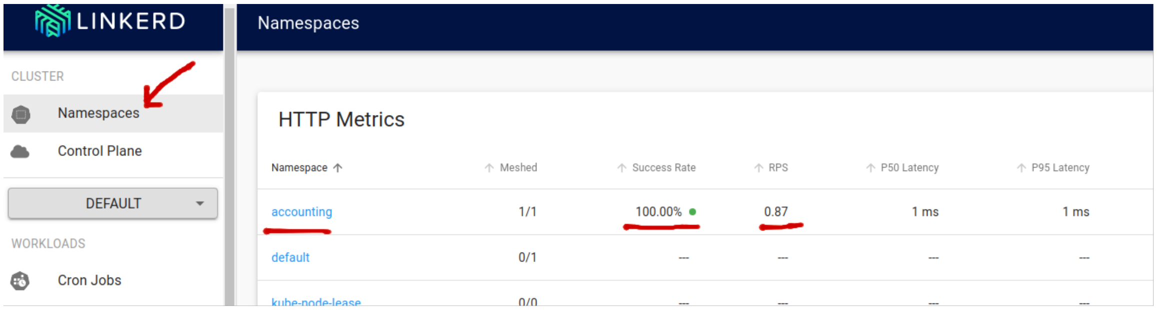


Figure 11.2: **Now shows meshed**

9. Scale up the `nginx-one` deployment. Generate traffic to get metrics for all the pods.

```
student@cp:~$ kubectl -n accounting scale deploy nginx-one --replicas=5
```

```
deployment.apps/nginx-one scaled
```

```
student@cp:~$ curl 10.102.8.205   #Several times
```

10. Explore some of the other information provided by the GUI. Note that the initial view is of the `default` namespaces. Change to `accounting` to see details of the nginx-one deployment.

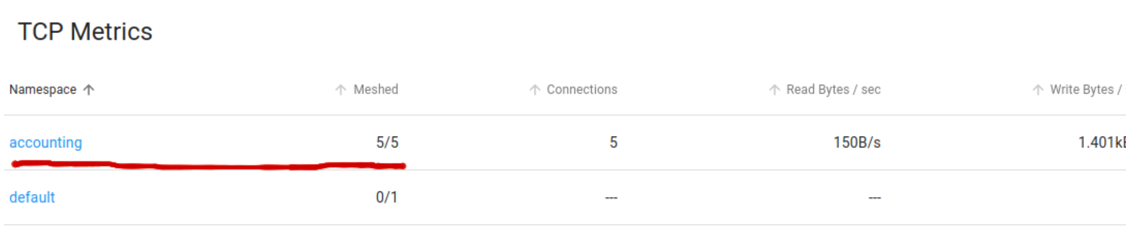


Figure 11.3: **Five meshed pods**

Exercise 11.2: Ingress Controller

We will use the **Helm** tool we learned about earlier to install an ingress controller.

1. Create two deployments, `web-one` and `web-two`, one running httpd, the other nginx. Expose both as ClusterIP services. Use previous content to determine the steps if you are unfamiliar. Test that both ClusterIPs work before continuing to the next step.

2. Linkerd does not come with an ingress controller, so we will add one to help manage traffic. We will leverage a **Helm** chart to install an ingress controller. Search the hub to find that there are many available.

```
student@cp:~$ helm search hub ingress
```

```
URL                                                          CHART VERSION
APP VERSION                DESCRIPTION
https://artifacthub.io/packages/helm/k8s-as-hel...          1.0.2
v1.0.0                     Helm Chart representing a single Ingress Kubern...
https://artifacthub.io/packages/helm/openstack-...          0.2.1
v0.32.0                    OpenStack-Helm Ingress Controller
<output_omitted>
```



```
https://artifacthub.io/packages/helm/api/ingres...        3.29.1
0.45.0                     Ingress controller for Kubernetes using NGINX a...
https://artifacthub.io/packages/helm/wener/ingr...        3.31.0
0.46.0                     Ingress controller for Kubernetes using NGINX a...
https://artifacthub.io/packages/helm/nginx/ngin...        0.9.2
1.11.2                     NGINX Ingress Controller
<output_omitted>
```

3. We will use a popular ingress controller provided by **NGINX**.

```
student@cp:~$ helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
```

```
"ingress-nginx" has been added to your repositories
```

```
student@cp:~$ helm repo update
```

```
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "ingress-nginx" chart repository
Update Complete. -Happy Helming!-
```

4. Download and edit the `values.yaml` file and change it to use a `DaemonSet` instead of a `Deployment`. This way there will be a pod on every node to handle traffic.

```
student@cp:~$ helm fetch ingress-nginx/ingress-nginx --untar
```

```
student@cp:~$ cd ingress-nginx
```

```
student@cp:~/ingress-nginx$ ls
```

```
CHANGELOG.md  Chart.yaml  OWNERS  README.md  ci  templates  values.yaml
```

```
student@cp:~/ingress-nginx$ vim values.yaml
```

values.yaml

```yaml
....
  ## DaemonSet or Deployment
  ##
  kind: DaemonSet                      #<-- Change to DaemonSet, around line 150

  ## Annotations to be added to the controller Deployment or DaemonSet
....
```

5. Now install the controller using the chart. Note the use of the dot (.) to look in the current directory.

```
student@cp:~/ingress-nginx$ helm install myingress .
```

```
NAME: myingress
LAST DEPLOYED: Thu Aug 23 13:47:16 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The ingress-nginx controller has been installed.
It may take a few minutes for the LoadBalancer IP to be available.
You can watch the status by running
'kubectl --namespace default get services -o wide -w myingress-ingress-nginx-controller'
```

```
An example Ingress that makes use of the controller:
<output_omitted>
```

6. We now have an ingress controller running, but no rules yet. View the resources that exist. Use the **-w** option to watch the ingress controller service show up. After it is available use **ctrl-c** to quit and move to the next command.

```
student@cp:~$ kubectl get ingress --all-namespaces
```

```
No resources found
```

```
student@cp:~$ kubectl --namespace default get services -o wide  myingress-ingress-nginx-controller
```

```
NAME                                   TYPE           CLUSTER-IP      EXTERNAL-IP
   PORT(S)                       AGE   SELECTOR
myingress-ingress-nginx-controller   LoadBalancer   10.104.227.79   <pending>
   80:32558/TCP,443:30219/TCP   47s   app.kubernetes.io/component=controller,
   app.kubernetes.io/instance=myingress,app.kubernetes.io/name=ingress-nginx
```

```
student@cp:~$ kubectl get pod --all-namespaces |grep nginx
```

```
default      myingress-ingress-nginx-controller-mrqt5    1/1     Running   0      20s
default      myingress-ingress-nginx-controller-pkdxm    1/1     Running   0      62s
default      nginx-b68dd9f75-h6ww7                       1/1     Running   0      21h
```

7. Now we can add rules which match HTTP headers to services.

```
student@cp:~$ vim ingress.yaml
```

ingress.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    nginx.ingress.kubernetes.io/service-upstream: "true"
  namespace: default
spec:
  ingressClassName: nginx
  rules:
  - host: www.external.com
    http:
      paths:
      - backend:
          service:
            name: secondapp
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
```

8. Create then verify the ingress is working. If you don't pass a matching header you should get a 404 error.

```
student@cp:~$ kubectl create -f ingress.yaml
```

```
ingress.networking.k8s.io/ingress-test created
```

```
student@cp:~$ kubectl get ingress
```

```
NAME           CLASS    HOSTS               ADDRESS   PORTS   AGE
ingress-test   nginx   www.external.com              80      5s
```

```
student@cp:~$ kubectl get pod -o wide |grep myingress
```

```
myingress-ingress-nginx-controller-mrqt5   1/1     Running   0          8m9s    192.168.219.118
  cp    <none>           <none>
myingress-ingress-nginx-controller-pkdxm   1/1     Running   0          8m9s    192.168.219.118
  cp    <none>           <none>
```

```
student@cp:~/ingress-nginx$ curl 192.168.219.118
```

```
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

9. Check the ingress service and expect another 404 error, don't use the admission controller.

```
student@cp:~/ingress-nginx$ kubectl get svc |grep ingress
```

```
myingress-ingress-nginx-controller             LoadBalancer   10.104.227.79    <pending>
  80:32558/TCP,443:30219/TCP   10m
myingress-ingress-nginx-controller-admission   ClusterIP      10.97.132.127    <none>
  443/TCP                      10m
```

```
student@cp:~/ingress-nginx$ curl 10.104.227.79
```

```
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

10. Now pass a header which matches a URL to one of the services we exposed in an earlier step. You should see the default nginx or httpd web server page.

```
student@cp:~/ingress-nginx$ curl -H "Host: www.external.com" http://10.104.227.79
```

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

11. We can add an annotation to the ingress pods for Linkerd. You will get some warnings, but the command will work.

```
student@cp:~/ingress-nginx$ kubectl get ds myingress-ingress-nginx-controller -o yaml  | \
    linkerd inject --ingress - | kubectl apply -f -
```

```
daemonset "myingress-ingress-nginx-controller" injected

Warning: resource daemonsets/myingress-ingress-nginx-controller is missing the
kubectl.kubernetes.io/last-applied-configuration annotation which is required
```

```
by kubectl apply. kubectl apply should only be used on resources created
declaratively by either kubectl create --save-config or kubectl apply. The
missing annotation will be patched automatically.
daemonset.apps/myingress-ingress-nginx-controller configured
```

12. Go to the `Top` page, change the namespace to default and the resource to `daemonset/myingress-ingress-nginx-controller`. Press start then pass more traffic to the ingress controller and view traffic metrics via the GUI. Let top run so we can see another page added in an upcoming step.

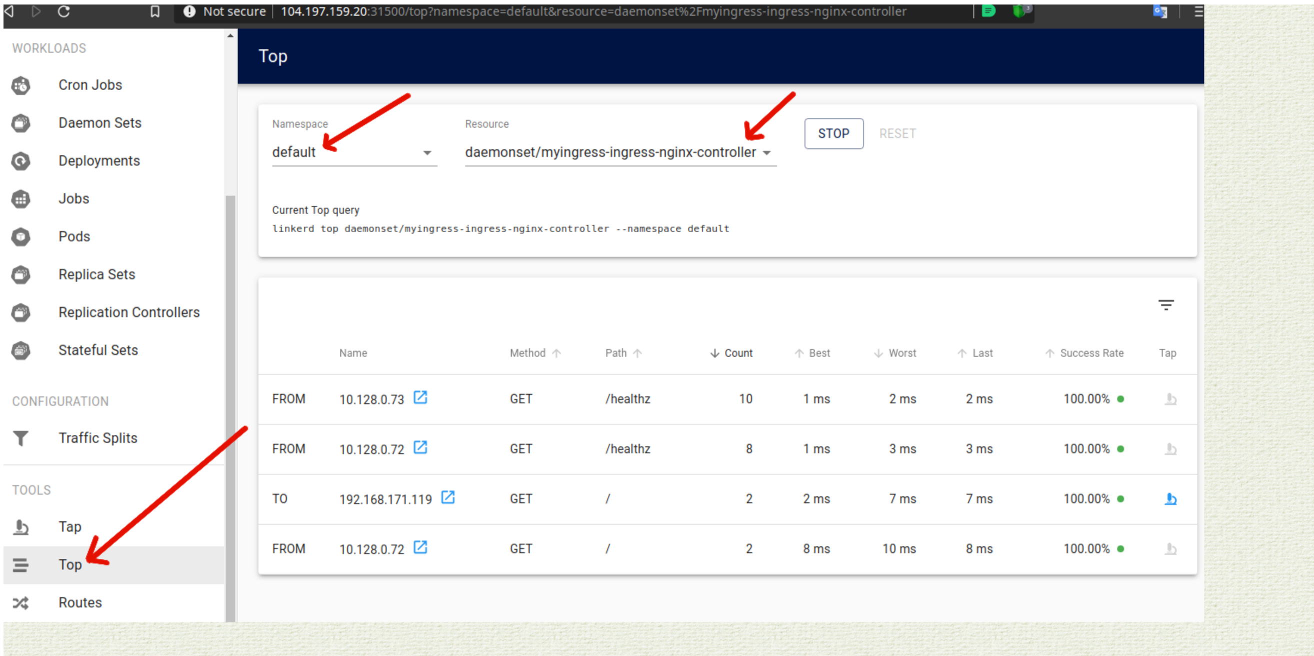


Figure 11.4: **Ingress Traffic**

13. At this point we would keep adding more and more servers. We'll configure one more, which would then could be a process continued as many times as desired.

    Customize the web-two (or whichever deployment is running nginx) welcome page. Run a bash shell inside the web-two pod. Your pod name will end differently. Install **vim** or an editor inside the container then edit the `index.html` file of nginx so that the title of the web page will be `Internal Welcome Page`. Much of the command output is not shown below.

```
student@cp:˜$ kubectl exec -it web-two-<Tab> -- /bin/bash
```

On Container

```
root@web-two-...-:/# apt-get update

root@web-two-...-:/# apt-get install vim -y

root@web-two-...-:/# vim /usr/share/nginx/html/index.html
```

```
<!DOCTYPE html>
<html>
<head>
<title>Internal Welcome Page</title>      #<-- Edit this line
<style>
<output_omitted>
```



```
root@thirdpage-:/$ exit
```

Edit the ingress rules to point the thirdpage service. It may be easiest to copy the existing `host` stanza and edit the `host` and `name`.

14. `student@cp:~$ kubectl edit ingress ingress-test`

ingress-test

```yaml
....
spec:
  rules:
  - host: internal.org
    http:
      paths:
      - backend:
          service:
            name: web-two
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
  - host: www.external.com
    http:
      paths:
      - backend:
          service:
            name: web-one
            port:
              number: 80
        path: /
        pathType: ImplementationSpecific
status:
....
```

15. Test the second `Host:` setting using **curl** locally as well as from a remote system, be sure the `<title>` shows the non-default page. Use the main IP of either node. The Linkerd GUI should show a new `TO` line, if you select the small blue box with an arrow you will see the traffic is going to internal.org.

```
student@cp:~$ curl -H "Host: internal.org" http://10.128.0.7/
```

```
<!DOCTYPE html>
<html>
<head>
<title>Internal Welcome Page</title>
<style>
<output_omitted>
```

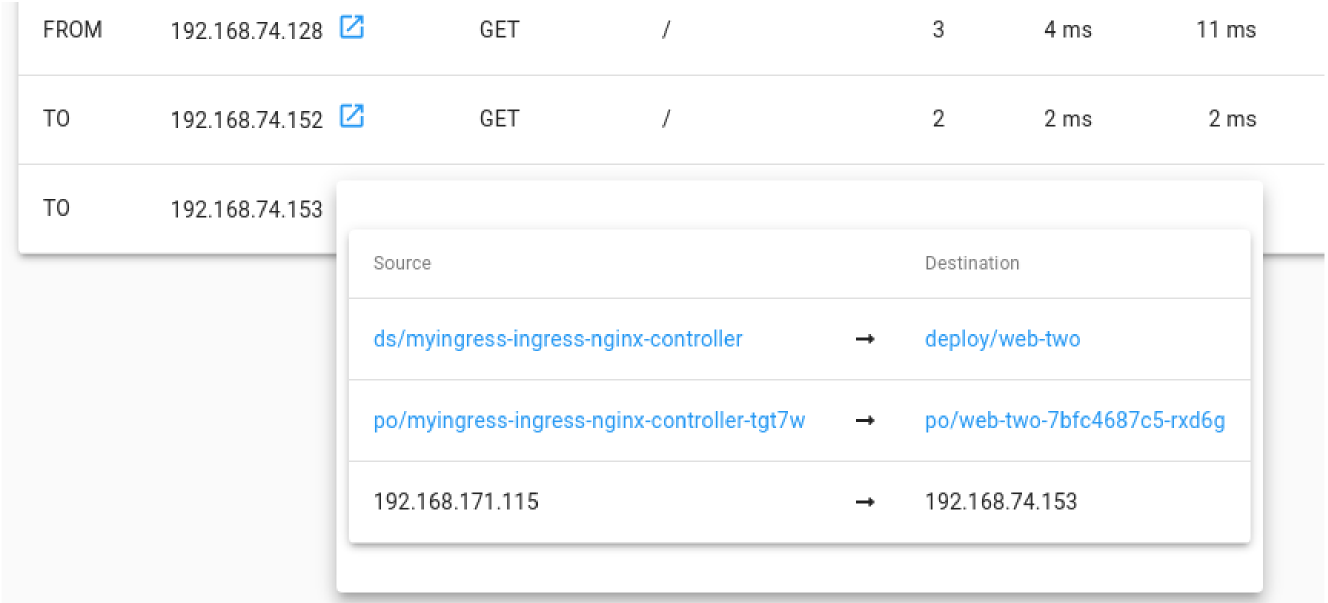


Figure 11.5: **Linkerd Top Metrics**

Chapter 12

Scheduling

12.1 Labs

Exercise 12.1: 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@cp:~$ kubectl get nodes
```

```
NAME      STATUS   ROLES           AGE    VERSION
cp        Ready    control-plane   44h    v1.28.1
worker    Ready    <none>          43h    v1.28.1
```

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

```
student@cp:~$ kubectl describe nodes |grep -A5 -i label
```

```
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/arch=amd64
                    kubernetes.io/hostname=scp
                    kubernetes.io/os=linux
                    node-role.kubernetes.io/control-plane=
--
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/arch=amd64
                    kubernetes.io/hostname=worker
```

```
                    kubernetes.io/os=linux
                    system=secondOne
```

```
student@cp:~$ kubectl describe nodes |grep -i taint
```

```
Taints:                         <none>
Taints:                         <none>
```

3. Get a count of how many containers are running on both the cp and worker nodes. There are about 24 containers running on the cp 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. Take note of the number of containers, and then notice the numbers change due to scheduling. The change between nodes is the important information, not the particular number. If you are using **cri-o** you can view containers using **crictl ps**.

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

```
NAMESPACE     NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
accounting    nginx-one                   1/1     1            1           19h
default       anotherweb-apache           1/1     1            1           8h
default       web-one                     1/1     1            1           45m
default       web-two                     1/1     1            1           45m
kube-system   cilium-operator             1/1     1            1           35h
<output_omitted>
```

```
student@cp:~$ sudo crictl ps | wc -l
```

```
24
```

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

```
21
```

4. For the purpose of the exercise we will assign the cp node to be VIP hardware and the secondary node to be for others.

```
student@cp:~$ kubectl label nodes cp status=vip
```

```
node/cp labeled
```

```
student@cp:~$ kubectl label nodes worker status=other
```

```
node/worker labeled
```

5. Verify your settings. You will also find there are some built in labels such as hostname, os and architecture type. The output below appears on multiple lines for readability.

```
student@cp:~$ kubectl get nodes --show-labels
```

```
NAME     STATUS   ROLES           AGE   VERSION   LABELS
cp       Ready    control-plane   35h   v1.28.1   beta.kubernetes.io/arch=amd64,
beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=cp,
kubernetes.io/os=linux,node-role.kubernetes.io/control-plane=,node-role.kubernetes.io/master=,
node.kubernetes.io/exclude-from-external-load-balancers=,status=vip
worker   Ready    <none>          35h   v1.28.1   beta.kubernetes.io/arch=amd64,
beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=worker,
kubernetes.io/os=linux,status=other,system=secondOne
```

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

```
student@cp:~$ vim vip.yaml
```

vip.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: vip
spec:
  containers:
  - name: vip1
    image: busybox
    args:
    - sleep
    - "1000000"
  - name: vip2
    image: busybox
    args:
    - sleep
    - "1000000"
  - name: vip3
    image: busybox
    args:
    - sleep
    - "1000000"
  - name: vip4
    image: busybox
    args:
    - sleep
    - "1000000"
  nodeSelector:
    status: vip
```

7. Deploy the new pod. Verify the containers have been created on the cp node. It may take a few seconds for all the containers to spawn. Check both the cp and the secondary nodes. From this point forward use **crictl** where the step lists **docker** if you have deployed your cluster with cri-o.

```
student@cp:~$ kubectl create -f vip.yaml
```

```
pod/vip created
```

```
student@cp:~$ sudo cat <<EOF | sudo tee /etc/crictl.yaml
runtime-endpoint: unix:///run/containerd/containerd.sock
image-endpoint: unix:///var/run/dockershim.sock
timeout: 10
debug: false
EOF
```

```
student@cp:~$ sudo crictl ps |wc -l
```

```
28
```

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

```
21
```

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@cp:~$ kubectl delete pod vip
```

```
pod "vip" deleted
```



```
student@cp:~$ vim vip.yaml
```

```
....
# nodeSelector:
 #    status: vip
```

9. Create the pod again. Containers can now be spawning on either of the node. You may see pods for the `daemonsets` as well.

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

```
<output_omitted>
```

```
student@cp:~$ 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. Again, the numbers may be different, the change in numbers is what we are looking for. Due to lack of `nodeSelector` they could go to either node.

```
student@cp:~$ sudo crictl ps |wc -l
```

```
24
```

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

```
25
```

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

```
student@cp:~$ cp vip.yaml other.yaml
```

```
student@cp:~$ sed -i s/vip/other/g other.yaml
```

```
student@cp:~$ vim other.yaml
```

YAML **other.yaml**

```yaml
....
  nodeSelector:
    status: other
```

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

```
student@cp:~$ kubectl create -f other.yaml
```

```
pod/other created
```

```
student@cp:~$ sudo crictl ps |wc -l
```

```
24
```

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

```
25
```


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

```
student@cp:~$ kubectl delete pods vip other
```

```
pod "vip" deleted
pod "other" deleted
```

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

```
<output_omitted>
```

Exercise 12.2: 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 cp node begins with a `NoSchedule` taint. We will work with three taints to limit or remove running pods.

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

```
student@cp:~$ vim taint.yaml
```

taint.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: taint-deployment
spec:
  replicas: 8
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name:  nginx
        image: nginx:1.20.1
        ports:
        - containerPort: 80
```

2. Apply the file to create the deployment.

```
student@cp:~$ kubectl apply -f taint.yaml
```

```
deployment.apps/taint-deployment created
```

3. Determine where the containers are running. In the following example three have been deployed on the cp node and five on the secondary node. Remember there will be other housekeeping containers created as well. Your numbers may be different, the actual number is not important, we are tracking the change in numbers.

```
student@cp:~$ sudo crictl ps |grep nginx
```

```
00c1be5df1e7        nginx@sha256:e3456c851a152494c3e.....
<output_omitted>
```

```
student@cp:~$ sudo crictl ps |wc -l
```

```
27
```

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

```
17
```

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

```
student@cp:~$ kubectl delete deployment taint-deployment
```

```
deployment.apps "taint-deployment" deleted
```

```
student@cp:~$ sudo crictl ps |wc -l
```

```
21
```

5. 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@cp:~$ kubectl taint nodes worker \
          bubba=value:PreferNoSchedule
```

```
node/worker tainted
```

```
student@cp:~$ kubectl describe node |grep Taint
```

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

```
student@cp:~$ kubectl apply -f taint.yaml
```

```
deployment.apps/taint-deployment created
```

6. Locate where the containers are running. We can see that more containers are on the cp, but there still were some created on the secondary. Delete the deployment when you have gathered the numbers.

```
student@cp:~$ sudo crictl ps |wc -l
```

```
21
```

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

```
23
```

```
student@cp:~$ kubectl delete deployment taint-deployment
```

```
deployment.apps "taint-deployment" deleted
```

7. Remove the taint, verify it has been removed. Note that the key is used with a minus sign appended to the end.

```
student@cp:˜$ kubectl taint nodes worker bubba-
```

```
node/worker untainted
```

```
student@cp:˜$ kubectl describe node |grep Taint
```

```
Taints:                        <none>
Taints:                        <none>
```

8. 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@cp:˜$ kubectl taint nodes worker \
          bubba=value:NoSchedule
```

```
node/worker tainted
```

```
student@cp:˜$ kubectl apply -f taint.yaml
```

```
deployment.apps/taint-deployment created
```

```
student@cp:˜$ sudo crictl ps |wc -l
```

```
21
```

```
student@worker:˜$ sudo crictl ps |wc -l
```

```
23
```

9. 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@cp:˜$ kubectl delete deployment taint-deployment
```

```
deployment.apps "taint-deployment" deleted
```

```
student@cp:˜$ kubectl taint nodes worker bubba-
```

```
node/worker untainted
```

```
student@cp:˜$ kubectl apply -f taint.yaml
```

```
deployment.apps/taint-deployment created
```

```
student@cp:˜$ sudo crictl ps |wc -l
```

```
27
```

```
student@worker:˜$ sudo crictl ps |wc -l
```

```
17
```

10. 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@cp:˜$ kubectl taint nodes worker \
         bubba=value:NoExecute
```

```
node "worker" tainted
```

student@cp:~$ sudo crictl ps |wc -l

```
37
```

student@worker:~$ sudo crictl ps |wc -l

```
5
```

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

student@cp:~$ kubectl taint nodes worker bubba-

```
node/worker untainted
```

student@cp:~$ sudo crictl ps |wc -l

```
32
```

student@worker:~$ sudo crictl ps |wc -l

```
6
```

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

student@cp:~$ kubectl delete deployment taint-deployment

```
deployment.apps "taint-deployment" deleted
```

13. **CHALLENGE STEP** Use your knowledge of deployments and scaling items to deploy multiple `httpd` pods across the nodes and examine the typical spread and spread after using taints.

Chapter 13

Logging and Troubleshooting

13.1 Labs

Exercise 13.1: 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@cp:~$ 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.

```
student@cp:~$ sudo find / -name "*apiserver*log"
```

```
/var/log/containers/kube-apiserver-cp_kube-system_kube-apiserver-423
d25701998f68b503e64d41dd786e657fc09504f13278044934d79a4019e3c.log
```

3. Take a look at the log file.

```
student@cp:~$ sudo less /var/log/containers/kube-apiserver-cp_kube-system_kube-
apiserver-423d25701998f68b503e64d41dd786e657fc09504f13278044934d79a4019e3c.log
```

```
<output_omitted>
```

4. Search for and review other log files for `coredns`, `kube-proxy`, and other cluster agents.

5. If **not** on a Kubernetes cluster using **systemd** which collects logs via **journalctl** you can view the text files on the cp 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 13.2: 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@cp:~$ kubectl get po --all-namespaces
```

```
NAMESPACE     NAME                                          READY     STATUS     RESTARTS   AGE
kube-system   cilium-operator-788c7d7585-jgn6s              1/1      Running    0          13m
kube-system   cilium-5tv9d                                  1/1      Running    0          6d1h

....
kube-system   etcd-cp                       1/1       Running   2          44h
kube-system   kube-apiserver-cp             1/1       Running   2          44h
kube-system   kube-controller-manager-cp    1/1       Running   2          44h
kube-system   kube-scheduler-cp             1/1       Running   2          44h
....
```

2. View the logs associated with various infrastructure pods. Using the **Tab** key you can get a list and choose a container. Then you can start typing the name of a pod and use **Tab** to complete the name.

```
student@cp:~$ kubectl -n kube-system logs <Tab><Tab>
```

```
cilium-operator-788c7d7585-jgn6s
cilium-5tv9d
coredns-5644d7b6d9-k7kts
coredns-5644d7b6d9-rnr2v
etcd-cp
kube-apiserver-cp
kube-controller-manager-cp
kube-proxy-qhc4f
kube-proxy-s56hl
kube-scheduler-f-cp
traefik-ingress-controller-hw5tv
traefik-ingress-controller-mcn47
```

```
student@cp:~$  kubectl -n kube-system logs \
          kube-apiserver-cp
```

```
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: v1.28.1
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>
```

3. View the logs of other Pods in your cluster.

Exercise 13.3: 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

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

```
student@cp:~$ git clone \
      https://github.com/kubernetes-incubator/metrics-server.git
```

```
<output_omitted>
```

2. As the software may have changed it is a good idea to read the **README.md** file for updated information.

```
student@cp:~$ cd metrics-server/ ; less README.md
```

```
<output_omitted>
```

3. Create the necessary objects. Be aware as new versions are released there may be some changes to the process and the created objects. Use the components.yaml to create the objects. The backslash is not necessary if you type it all on one line.

```
student@cp:~$ kubectl create -f \
https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

```
serviceaccount/metrics-server created
clusterrole.rbac.authorization.k8s.io/system:aggregated-metrics-reader created
clusterrole.rbac.authorization.k8s.io/system:metrics-server created
rolebinding.rbac.authorization.k8s.io/metrics-server-auth-reader created
clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-delegator created
clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server created
service/metrics-server created
deployment.apps/metrics-server created
apiservice.apiregistration.k8s.io/v1beta1.metrics.k8s.io created
```

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

```
student@cp:~$ 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
```

5. Edit the `metrics-server` deployment to allow insecure TLS. The default certificate is x509 self-signed and not trusted by default. In production you may want to configure and replace the certificate. You may encounter other issues as this software is fast-changing. The need for the `kubelet-preferred-address-types` line has been reported on some platforms.

```
student@cp:~$ kubectl -n kube-system edit deployment metrics-server
```

```yaml
....
    spec:
      containers:
      - args:
        - --cert-dir=/tmp
        - --secure-port=4443
        - --kubelet-insecure-tls                    #<-- Add this line
        - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname #<--May be needed
        image: k8s.gcr.io/metrics-server/metrics-server:v0.3.7
....
```

6. Test that the metrics server pod is running and does not show errors. At first you should see a few lines showing the container is listening. As the software changes these messages may be slightly different.

```
student@cp:~$ kubectl -n kube-system logs metrics-server<TAB>
```

```
I0207 14:08:13.383209       1 serving.go:312] Generated self-signed cert
(/tmp/apiserver.crt, /tmp/apiserver.key)
I0207 14:08:14.078360       1 secure_serving.go:116] Serving securely on
[::]:4443
```

7. Test that the metrics working by viewing pod and node metrics. Your output may have different pods. It can take an minute or so for the metrics to populate and not return an error.

```
student@cp:~$ sleep 120 ; kubectl top pod --all-namespaces
```

```
NAMESPACE     NAME                                      CPU(cores)   MEMORY(bytes)
kube-system   cilium-kube-controllers-7b9dcdcc5-qg6zd   2m           6Mi
kube-system   cilium-node-dr279                         23m          22Mi
kube-system   cilium-node-xtvfd                         21m          22Mi
kube-system   coredns-5644d7b6d9-k7kts                  2m           6Mi
```

```
kube-system   coredns-5644d7b6d9-rnr2v                  3m           6Mi
<output_omitted>
```

```
student@cp:~$ kubectl top nodes
```

```
NAME                CPU(cores)    CPU%    MEMORY(bytes)    MEMORY%
cp    228m         11%     2357Mi          31%
worker      76m           3%      1385Mi           18%
```

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

```
student@cp:~$ curl --cert ./client.pem \
    --key ./client-key.pem --cacert ./ca.pem \
    https://k8scp: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": "2023-08-10T20:27:00Z"
      },
      "timestamp": "2023-08-10T20:26:18Z",
      "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 full wanted functionality.

The first commands do not have the details. Refer to earlier content as necessary.

1. Search https://artifacthub.io/ for the `helm` organization and the `kubernetes-dashboard` chart.

2. Fetch the chart and edit the `values.yaml` file.

```yaml
....
service:
  type: NodePort          #<-- Change to NodePort
  externalPort: 443
....
```

3. Install the chart and give it a name of `dashboard`

4. The helm chart version does not allow any resource access by default. We will give the dashboard full admin rights, which may be more than one would in production. The dashboard is running in the `default` namespace. First find the name of the service account, which is based off the name you used for the chart. Note: In few dashboard chart, the dashboard might be deployed in `kubernetes-dashboard` namespace.

   There is more on service account in the `Security` chapter.

```
student@cp:~$ kubectl get serviceaccounts
```

```
NAME                            SECRETS   AGE
dashboard-kubernetes-dashboard  1         6m
default                         1         2d21h
myingress-ingress-nginx         1         42h
```

```
student@cp:~$ kubectl create clusterrolebinding dashaccess \
        --clusterrole=cluster-admin \
        --serviceaccount=default:kubernetes-dashboard
```

```
clusterrolebinding.rbac.authorization.k8s.io/dashaccess created
```

5. On your local system 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**. Some browsers won't even give you to option. If nothing shows up try a different browser. The page should then show the `Kubernetes Dashboard`. You may be able to find the public IP address using **curl**.

```
student@cp:~$ curl ifconfig.io
```

```
35.231.8.178
```

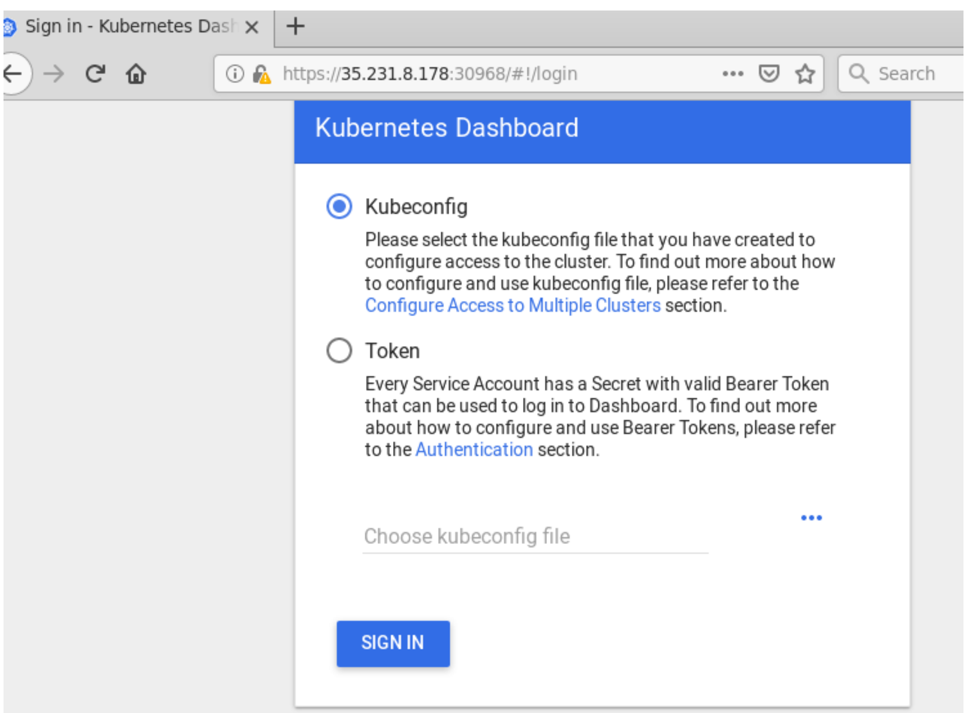


Figure 13.1: **External Access via Browser**

6. 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@cp:~$ kubectl create token kubernetes-dashboard
```


```
eyJlxvezoLAilithbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZX
JuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY
291bnQvc2VjcmV0Lm5hbWUiOiJrdWJlcm5ldGVzLWRhc2hib2FyZC10b2tlbi1wbW04NCIsImt1YmVybmV0ZXMuaW8vc2Vydmlj
ZWFjY291bnQvc2VydmljZS1hY2NvdW50Lm5hbWUiOiJrdWJlcm5ldGVzLWRhc2hib2FyZCIsImt1YmVybmV0ZXMuaW8vc2Vydml
jZWFjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6IjE5MDY4ZDIzLTE1MTctMTFlOS1hZmMyLTQyMDEwYThlMDAwMyIsInN1Yi
I6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDprdWJlLXN5c3RlbTprdWJlcm5ldGVzLWRhc2hib2FyZCJ9.aYTUMWr290pjt5i32rb8
qXpq4onn3hLhvz6yLSYexgRd6NYsygVUyqnkRsFE1trg9i1ftNXKJdzkY5kQzN3AcpUTvyj_BvJgzNh3JM9p7QMjI8LHTz4TrRZ
rvwJVWitrEn4VnTQuFVcADFD_rKB9FyI_gvT_QiW5fQm24ygTIgf0Yd44263oakG8sL64q7UfQNW2wt5SOorMUtybOmX4CXNUYM8
G44ejEtv9GW5OsVjEmLIGaoEMX7fctwUN_XCyPdzcCg2WOxRHahBJmbCuLz2SSWL52q4nXQmhTq_L8VDDpt6LjEqXW6LtDJZGjVC
s2MnBLerQz-ZAgsVaubbQ
```

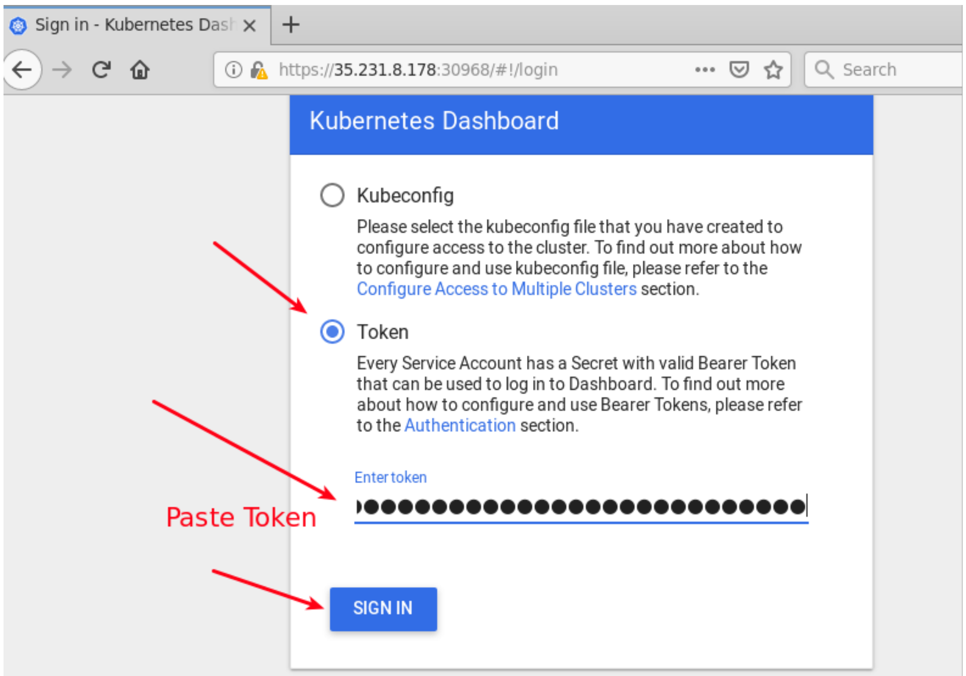



Figure 13.2: **External Access via Browser**

7. Navigate around the various sections and use the menu to the left as time allows. As the pod view is of the `default` namespace, you may want to switch over to the `kube-system` namespace or create a new deployment to view the resources via the GUI. Scale the deployment up and down and watch the responsiveness of the GUI.

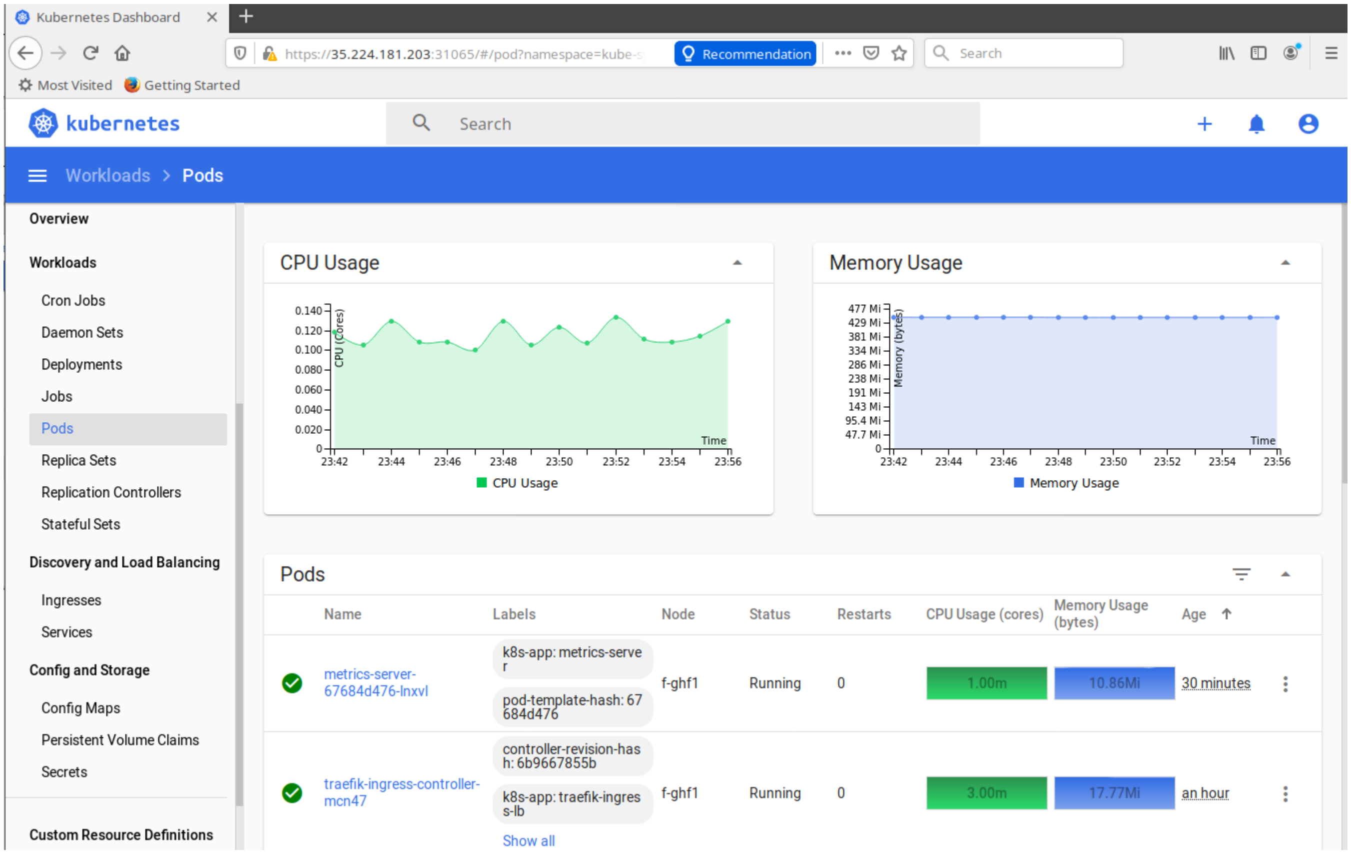


Figure 13.3: **External Access via Browser**

Chapter 14

Custom Resource Definition

14.1 Labs

Exercise 14.1: Create a Custom Resource Definition

Overview

The use of `CustomResourceDefinitions (CRD)`, has become a common manner to deploy new objects and operators. Creation of a new operator is beyond the scope of this course, basically it is a watch-loop comparing a spec to the current status, and making changes until the states match. A good discussion of creating a operators can be found here: https://operatorframework.io/.

First we will examine an existing CRD, then make a simple CRD, but without any particular action. It will be enough to find the object ingested into the API and responding to commands.

1. View the existing CRDs.

```
student@cp:~$ kubectl get crd --all-namespaces
```

```
 NAME                                            CREATED AT
NAME                                            CREATED AT
authorizationpolicies.policy.linkerd.io         2023-08-28T11:30:34Z
ciliumcidrgroups.cilium.io                      2023-08-28T08:58:54Z
ciliumclusterwidenetworkpolicies.cilium.io      2023-08-28T08:58:57Z
<output_omitted>
```

2. We can see from the names that these CRDs are all working on `Cilium`, our network plugin. View the `cilium-cni.yaml` file we used when we initialized the cluster to see how these objects were created, and some CRD templates to review.

```
student@cp:~$ less cilium-cni.yaml
student@cp:~$ kubectl describe crd ciliumcidrgroups.cilium.io
```

```
<output_omitted>
---
Name:         ciliumcidrgroups.cilium.io
Namespace:
Labels:       io.cilium.k8s.crd.schema.version=1.26.10
```

```
Annotations:  <none>
API Version:  apiextensions.k8s.io/v1
Kind:         CustomResourceDefinition
Metadata:

<output_omitted>
```

3. Now that we have seen some examples, we will create a new YAML file.

```
student@cp:˜$ vim crd.yaml
```

crd.yaml

```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  # name must match the spec fields below, and be in the form: <plural>.<group>
  name: crontabs.stable.example.com
spec:
  # group name to use for REST API: /apis/<group>/<version>
  group: stable.example.com
  # list of versions supported by this CustomResourceDefinition
  versions:
    - name: v1
      # Each version can be enabled/disabled by Served flag.
      served: true
      # One and only one version must be marked as the storage version.
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                cronSpec:
                  type: string
                image:
                  type: string
                replicas:
                  type: integer
  # either Namespaced or Cluster
  scope: Namespaced
  names:
    # plural name to be used in the URL: /apis/<group>/<version>/<plural>
    plural: crontabs
    # singular name to be used as an alias on the CLI and for display
    singular: crontab
    # kind is normally the CamelCased singular type. Your resource manifests use this.
    kind: CronTab
    # shortNames allow shorter string to match your resource on the CLI
    shortNames:
    - ct
```

4. Add the new resource to the cluster.

```
student@cp:˜$ kubectl create -f crd.yaml
```

```
customresourcedefinition.apiextensions.k8s.io/crontabs.stable.example.com created
```

5. View and describe the resource. The new line may be in the middle of the output. You'll note the **describe** output is unlike other objects we have seen so far.

```
student@cp:~$ kubectl get crd
```

```
NAME                                    CREATED AT
<output_omitted>
crontabs.stable.example.com             2023-08-13T03:18:07Z
<output_omitted>
```

```
student@cp:~$ kubectl describe crd crontab<Tab>
```

```
Name:          crontabs.stable.example.com
Namespace:
Labels:        <none>
Annotations:   <none>
API Version:   apiextensions.k8s.io/v1
Kind:          CustomResourceDefinition
<output_omitted>
```

6. Now that we have a new API resource we can create a new object of that type. In this case it will be a crontab-like image, which does not actually exist, but is being used for demonstration.

```
student@cp:~$ vim new-crontab.yaml
```

YAML **new-crontab.yaml**

```yaml
apiVersion: "stable.example.com/v1"
    # This is from the group and version of new CRD
kind: CronTab
    # The kind from the new CRD
metadata:
  name: new-cron-object
spec:
  cronSpec: "*/5 * * * *"
  image: some-cron-image
    #Does not exist
```

7. Create the new object and view the resource using short and long name.

```
student@cp:~$ kubectl create -f new-crontab.yaml
```

```
crontab.example.com/new-cron-object created
```

```
student@cp:~$ kubectl get CronTab
```

```
NAME               AGE
new-cron-object    22s
```

```
student@cp:~$ kubectl get ct
```

```
NAME               AGE
new-cron-object    29s
```

```
student@cp:~$ kubectl describe ct
```

```
Name:          new-cron-object
Namespace:     default
Labels:        <none>
```



```
Annotations:  <none>
API Version:  stable.example.com/v1
Kind:         CronTab

<output_omitted>

Spec:
  Cron Spec:  */5 * * * *
  Image:      some-cron-image
Events:       <none>
```

8. To clean up the resources we will delete the CRD. This should delete all of the endpoints and objects using it as well.

```
student@cp:~$ kubectl delete -f crd.yaml
```

```
customresourcedefinition.apiextensions.k8s.io "crontabs.stable.example.com" deleted
```

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 cp 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@cp:~$ 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. Look at the status output. Follow the CGroup and kubelet information, which is a long line where configuration settings are drawn from, to find where the configuration file can be found.

   ```
   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@cp:~$ sudo less /var/lib/kubelet/config.yaml
```

config.yaml

```
<output_omitted>
rotateCertificates: true
runtimeRequestTimeout: 0s
shutdownGracePeriod: 0s
shutdownGracePeriodCriticalPods: 0s
staticPodPath: /etc/kubernetes/manifests
streamingConnectionIdleTimeout: 0s
syncFrequency: 0s
volumeStatsAggPeriod: 0s
```

4. Other agents on the cp 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@cp:~$ sudo ls /etc/kubernetes/manifests/
```

```
etcd.yaml            kube-controller-manager.yaml
kube-apiserver.yaml  kube-scheduler.yaml
```

```
student@cp:~$ 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@cp:~$  kubectl -n kube-system get secrets
```

```
NAME                                 TYPE
 DATA      AGE
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 `bootstrap-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@cp:~$ kubectl -n kube-system get secrets bootstrap-token<Tab> -o yaml
```


```
apiVersion: v1
data:
  auth-extra-groups: c3lzdGVtOmJvb3RzdHJhcHBlcnM6a3ViZWFkbTpkZWZhdWx0LW5vZGUtdG9rZW4=
  expiration: MjAyMy0wNi0wM1QxMjowOTowMlo=
  token-id: NXBvMGo3
  token-secret: MXhzMDgx0G1rcTFyeDQxbg==
  usage-bootstrap-authentication: dHJ1ZQ==
  usage-bootstrap-signing: dHJ1ZQ==
kind: Secret
metadata:
  creationTimestamp: "2023-08-01T12:09:02Z"
```

```yaml
  name: bootstrap-token-5po0j7
  namespace: kube-system
  resourceVersion: "209"
  uid: 98219199-4876-4cfa-a4be-586cda27cc6b
type: bootstrap.kubernetes.io/token
```

7. The **kubectl config** command can also be used to view and update parameters. When making updates this could avoid a typo removing access to the cluster. View the current configuration settings. The keys and certs are redacted from the output automatically.

```
student@cp:~$ 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@cp:~$ 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@cp:~$ cp $HOME/.kube/config $HOME/cluster-api-config
```

10. Explore working with cluster and security configurations both using **kubectl** and **kubeadm**. Among other values, find the name of your cluster. You will need to become `root` to work with **kubeadm**.

```
student@cp:~$ 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@cp:~$ sudo kubeadm token -h
```

```
<output_omitted>
```

```
student@cp:~$ 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@cp:~$ 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@cp:~$ kubectl create ns development
```

```
namespace/development created
```

```
student@cp:~$ 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@cp:~$ kubectl config get-contexts
```

```
CURRENT   NAME                          CLUSTER      AUTHINFO           NAMESPACE
*         kubernetes-admin@kubernetes   kubernetes   kubernetes-admin
```

3. Create a new user `DevDan` and assign a password of `lftr@in`.

```
student@cp:~$ sudo useradd -s /bin/bash DevDan
```

```
student@cp:~$ sudo passwd DevDan
```

```
Enter new UNIX password:  lftr@in
Retype new UNIX password:  lftr@in
passwd: password updated successfully
```

4. Generate a private key then Certificate Signing Request (CSR) for `DevDan`. On some Ubuntu 18.04 nodes a missing file may cause an error with random number generation. The **touch** command should ensure one way of success.

```
student@cp:~$ openssl genrsa -out DevDan.key 2048
```

```
Generating RSA private key, 2048 bit long modulus
......+++
.........+++
e is 65537 (0x10001)
```

```
student@cp:~$ touch $HOME/.rnd
```

```
student@cp:~$ openssl req -new -key DevDan.key \
      -out DevDan.csr -subj "/CN=DevDan/O=development"
```

5. Using thew newly created request generate a self-signed certificate using the x509 protocol. Use the CA keys for the Kubernetes cluster and set a 45 day expiration. You'll need to use **sudo** to access to the inbound files.

```
student@cp:~$ 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@cp:~$ 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@cp:~$ 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@cp:~$ 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@cp:~$ 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@cp:~$ 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@cp:~$ 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@cp:~$ vim role-dev.yaml
```

role-dev.yaml

```yaml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: development
  name: developer
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["deployments", "replicasets", "pods"]
  verbs: ["list", "get", "watch", "create", "update", "patch", "delete"]
# You can use ["*"] for all verbs
```

13. Create the object. Check white space and for typos if you encounter errors.

```
student@cp:~$ 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@cp:~$ vim rolebind.yaml
```

rolebind.yaml

```yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: developer-role-binding
  namespace: development
subjects:
- kind: User
  name: DevDan
  apiGroup: ""
roleRef:
  kind: Role
  name: developer
  apiGroup: ""
```

```
student@cp:~$ kubectl create -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@cp:~$  kubectl --context=DevDan-context get pods
```

```
No resources found in development namespace.
```

16. Create a new pod, verify it exists, then delete it.

```
student@cp:~$ kubectl --context=DevDan-context \
     create deployment  nginx --image=nginx
```

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

```
student@cp:~$ kubectl --context=DevDan-context get pods
```

```
NAME                     READY   STATUS    RESTARTS   AGE
nginx-7c87f569d-7gb9k    1/1     Running   0          5s
```

```
student@cp:~$ 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@cp:~$ cp role-dev.yaml role-prod.yaml
```

```
student@cp:~$ vim role-prod.yaml
```

role-prod.yaml

```yaml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: production       #<<- This line
  name: dev-prod              #<<- and this line
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["deployments", "replicasets", "pods"]
  verbs: ["get", "list", "watch"] #<<- and this one
```

```
student@cp:~$ cp rolebind.yaml rolebindprod.yaml
```

```
student@cp:~$ vim rolebindprod.yaml
```

rolebindprod.yaml

```yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: production-role-binding  #<-- Edit to production
  namespace: production          #<-- Also here
subjects:
- kind: User
  name: DevDan
  apiGroup: ""
roleRef:
  kind: Role
  name: dev-prod                 #<-- Also this
  apiGroup: ""
```

18. Create both new objects.

```
student@cp:~$ kubectl create -f role-prod.yaml
```

```
role.rbac.authorization.k8s.io/dev-prod created
```

```
student@cp:~$ kubectl create -f rolebindprod.yaml
```

```
rolebinding.rbac.authorization.k8s.io/production-role-binding created
```

19. Create the new context for production use.

```
student@cp:~$ 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@cp:~$ 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@cp:~$ 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@cp:~$ 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.

24. **OPTIONAL CHALLENGE STEP:** Become the `DevDan` user. Solve any missing configuration errors. Try to create a deployment in the `development` and the `production` namespaces. Do the errors look the same? Configure as necessary to only have two contexts available to DevDan.

```
DevDan@cp:~$ kubectl config get-contexts
```

```
CURRENT   NAME              CLUSTER      AUTHINFO   NAMESPACE
*         DevDan-context    kubernetes   DevDan     development
          ProdDan-context   kubernetes   DevDan     production
```

Exercise 15.3: Admission Controllers

The last stop before a request is sent to the API server is an `admission control plug-in`. They interact with features such as setting parameters like a default storage class, checking resource quotas, or security settings. A newer feature (v1.7.x) is dynamic controllers which allow new controllers to be ingested or configured at runtime.

1. View the current `admission controller` settings. Unlike earlier versions of Kubernetes the controllers are now compiled into the server, instead of being passed at run-time. Instead of a list of which controllers to use we can enable and disable specific plugins.

```
student@cp:~$ 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 cp nodes for quorum. Log into each and use the **ip** command to fill in the table with the IP addresses of the primary interface of each node. If using **GCE** nodes it would be `ens4`, yours may be different. You may need to install software such an editor on the nodes.

Proxy Node	
Second Control Plane	
Third Control Plane	

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.

High level steps:

1. Deploy a load balancer configured to pass through traffic on your new proxy node. HAProxy is easy to deploy using online documentation. Start with forwarding traffic of the cp alias to just the working cp.
2. Install the Kubernetes software on the second and third cp nodes.
3. Use **kubeadm join** on the second cp, adding it to the cluster as another control plane using the node name.
4. Join the third cp as another control plane to the cluster using the node name.

5. Update the proxy to use all three cps backend IPs.
6. Temporarily shut down the first cp and monitor traffic.

Exercise 16.2: Detailed Steps

Deploy a Load Balancer

While there are many options, both software and hardware, we will be using an open source tool **HAProxy** to configure a load balancer.

1. Deploy HAProxy. Log into the proxy node. Update the repos then install a the HAProxy software. Answer `yes`, should you the installation ask if you will allow services to restart.

```
student@ha-proxy:~$ sudo apt-get update ; sudo apt-get install -y haproxy vim
```

```
<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 cp node until we are sure the proxy is forwarding traffic to the known working cp.

```
student@ha-proxy:~$ sudo vim /etc/haproxy/haproxy.cfg
```

```
....
defaults
        log global                  #<-- Edit these three lines, starting around line 23
        option tcplog
        mode tcp
....
        errorfile 503 /etc/haproxy/errors/503.http
        errorfile 504 /etc/haproxy/errors/504.http

frontend proxynode                       #<-- Add the following lines to bottom of file
   bind *:80
   bind *:6443
   stats uri /proxystats
   default_backend k8sServers

backend k8sServers
   balance roundrobin
   server cp  10.128.0.24:6443 check  #<-- Edit these with your IP addresses, port, and hostname
#   server secondcp  10.128.0.30:6443 check  #<-- Comment out until ready
#   server thirdcp  10.128.0.66:6443 check   #<-- Comment out until ready
listen stats
   bind :9999
   mode http
   stats enable
   stats hide-version
   stats uri /stats
```

3. Restart the `haproxy` service and check the status. You should see the frontend and backend proxies report being started.

```
student@ha-proxy:~$ sudo systemctl restart haproxy.service
student@ha-proxy:~$ sudo systemctl status haproxy.service
```

```
<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 cp** Edit the `/etc/hosts` file and comment out the old and add a new `k8scp` alias to the IP address of the proxy server.

```
student@cp:~$ sudo vim /etc/hosts
```

```
10.128.0.64 k8scp     #<-- Add alias to proxy IP
#10.128.0.24 k8scp    #<-- Comment out the old alias, in case its needed
127.0.0.1 localhost
....
```

5. Use a local browser to navigate to the public IP of your proxy server. The http://34.69.XX.YY:9999/stats is an example your IP address would be different. Leave the browser up and refresh as you run following steps. You can find your public ip using **curl**. Your IP will be different than the one shown below.

```
ha-proxy$ curl ifconfig.io
```

```
34.69.73.159
```

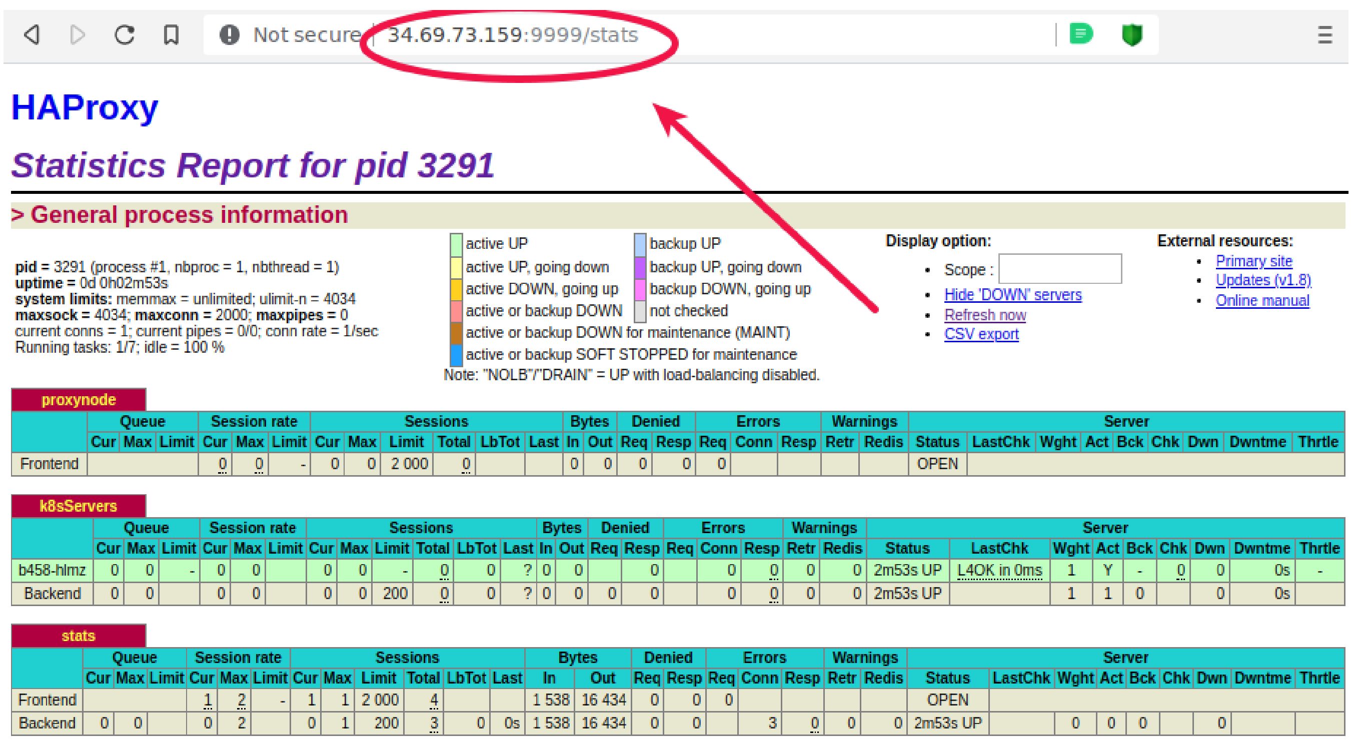


Figure 16.1: **Initial HAProxy Status**

6. Check the node status from the cp node then check the proxy statistics. You should see the byte traffic counter increase.

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

```
NAME     STATUS   ROLES           AGE    VERSION
cp       Ready    control-plane   2d6h   v1.28.1
worker   Ready    <none>          2d3h   v1.28.1
```

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 cp before adding the third cp

1. Configure and install the kubernetes software on the **second cp**. Use the same steps as when we first set up the cluster, earlier in the course. You may want to copy and paste from earlier commands in your **history** to make these steps easier. **All the steps up to but not including kubeadm init or kubeadm join** A script `k8sWorker.sh` has been included in the course tarball to make this process go faster, if you would like. View and edit the script to be the correct version before running it.

2. Install the software on the **third cp** using the same commands.

Join Control Plane Nodes

1. Edit the `/etc/hosts` file **ON ALL NODES** to ensure the alias of `k8scp` is set on each node to the proxy IP address. Your IP address may be different.

```
student@cp:~$ sudo vim /etc/hosts
```

```
10.128.0.64 k8scp
#10.128.0.24 k8scp
127.0.0.1 localhost
....
```

2. On the **first cp** 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@cp:~$ sudo kubeadm token create
```

```
jasg79.fdh4p2791320cz1g
```

4. Create a new SSL hash.

```
student@cp:~$ 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 cp certificate to join as a cp instead of as a worker.

```
student@cp:~$ 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 cp** use the previous output to build a **kubeadm join** command. Please be aware that multi-line copy and paste from Windows and some MacOS has paste issues. If you get unexpected output copy one line at a time.

```
student@$secondcp:~$ sudo kubeadm join k8scp:6443 \
 --token jasg79.fdh4p279l320cz1g \
 --discovery-token-ca-cert-hash sha256:f62bf97d4fba6876e4c3ff645df3fca969c06169dee3865aab9d0bca8ec9f8cd \
 --control-plane --certificate-key \
 5610b6f73593049acddee6b59994360aa4441be0c0d9277c76705d129ba18d65
```

```
[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/
<output_omitted>
```

7. Return to the first cp node and check to see if the node has been added and is listed as a `cp`.

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

```
NAME          STATUS   ROLES             AGE    VERSION
cp            Ready    control-plane     2d6h   v1.28.1
secondcp      Ready    control-plane     10m    v1.28.1
worker        Ready    <none>            2d3h   v1.28.1
```

8. Copy and paste the **kubeadm join** command to the third cp. Then check that the third cp has been added.

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

```
NAME       STATUS   ROLES            AGE    VERSION
cp         Ready    control-plane    2d6h   v1.28.1
secondcp   Ready    control-plane    13m    v1.28.1
thirdcp    Ready    control-plane    3m     v1.28.1
worker     Ready    <none>           2d3h   v1.28.1
```

9. Copy over the configuration file as suggested in the output at the end of the join command. Do this on both newly added cp nodes.

```
student@secondcp:~$ mkdir -p $HOME/.kube
student@secondcp:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
student@secondcp:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

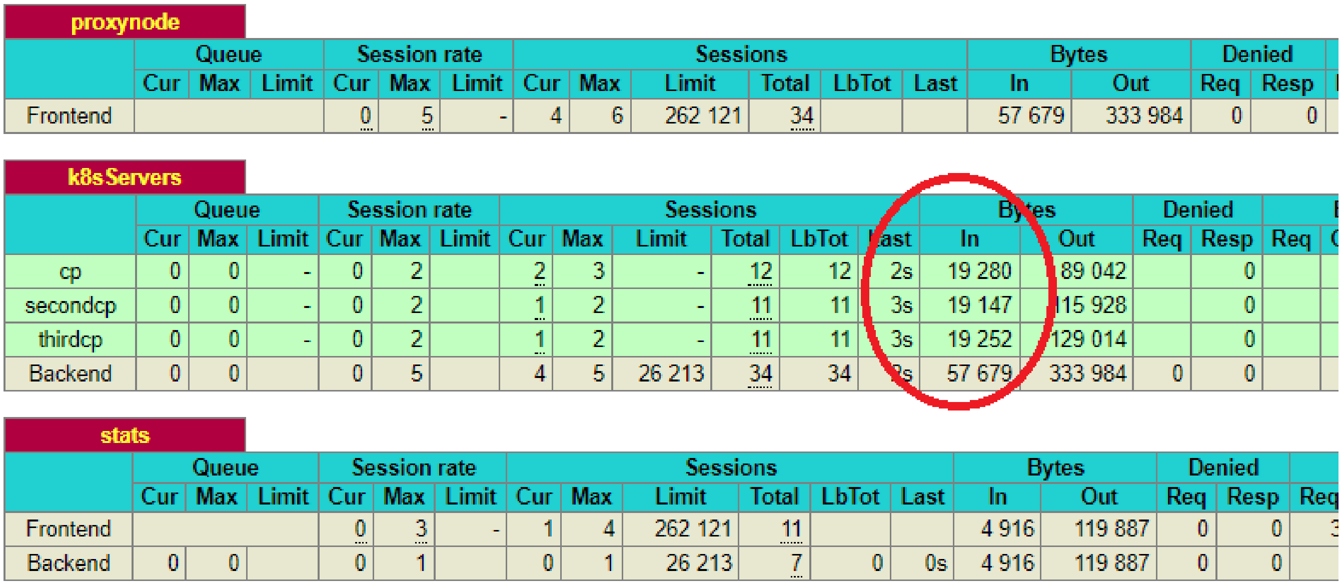
10. On the **Proxy node**. Edit the proxy to include all three cp nodes then restart the proxy.

```
student@ha-proxy:~$ sudo vim /etc/haproxy/haproxy.cfg
```

```
....
backend k8sServers
   balance roundrobin
   server cp        10.128.0.24:6443 check
   server secondcp  10.128.0.30:6443 check  #<-- Edit/Uncomment these lines
   server thirdcp   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	5	-	4	6	262 121	34			57 679	333 984	0	0

k8sServers

	Queue			Session rate			Sessions						Bytes		Denied		
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req
cp	0	0	-	0	2		2	3	-	12	12	2s	19 280	89 042		0	
secondcp	0	0	-	0	2		1	2	-	11	11	3s	19 147	115 928		0	
thirdcp	0	0	-	0	2		1	2	-	11	11	3s	19 252	129 014		0	
Backend	0	0		0	5		4	5	26 213	34	34	[illegible]s	57 679	333 984	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	Req
Frontend				0	3	-	1	4	262 121	11			4 916	119 887	0	0	3
Backend	0	0		0	1		0	1	26 213	7	0	0s	4 916	119 887	0	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 cp.

```
student@cp:~$ kubectl -n kube-system get pods |grep etcd
```

```
etcd-cp              1/1     Running    0          2d12h
etcd-secondcp        1/1     Running    0          22m
etcd-thirdcp         1/1     Running    0          18m
```

```
student@cp:~$ kubectl -n kube-system logs -f etcd-thirdcp
```

```
....
2023-08-09 01:58:03.768858 I | mvcc: store.index: compact 300473
2023-08-09 01:58:03.770773 I | mvcc: finished scheduled compaction at 300473 (took 1.286565ms)
2023-08-09 02:03:03.766253 I | mvcc: store.index: compact 301003
2023-08-09 02:03:03.767582 I | mvcc: finished scheduled compaction at 301003 (took 995.775µs)
2023-08-09 02:08:03.785807 I | mvcc: store.index: compact 301533
2023-08-09 02:08:03.787058 I | mvcc: finished scheduled compaction at 301533 (took 913.185µs)
```

13. Log into one of the **etcd** pods and check the cluster status, using the IP address of each server and port 2379. Your IP addresses may be different. Exit back to the node when done.

```
student@cp:~$ kubectl -n kube-system exec -it etcd-cp -- /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 |  3.5.7  |   24 MB |      true |        11 |
↪  392573 |
| 10.128.0.24:2379 | d2620a7d27a9b449 |  3.5.7  |   24 MB |     false |        11 |
↪  392573 |
| 10.128.0.30:2379 | ef44cc541c5f37c7 |  3.5.7  |   24 MB |     false |        11 |
↪  392573 |
+------------------+------------------+---------+---------+-----------+-----------+------------+
```

Test Failover

Now that the cluster is running and has chosen a leader we will shut down containerd, 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 service on the node which shows `IS LEADER` set to `true`.

   ```
   student@cp:~$ sudo systemctl stop containerd.service
   ```

   If you chose cri-o as the container engine then the cri-o service and conmon processes are distinct. It may be easier to reboot the node and refresh the HAProxy web page until it shows the node is down. It may take a while for the node to finish the boot process. The second and third cp should work the entire time.

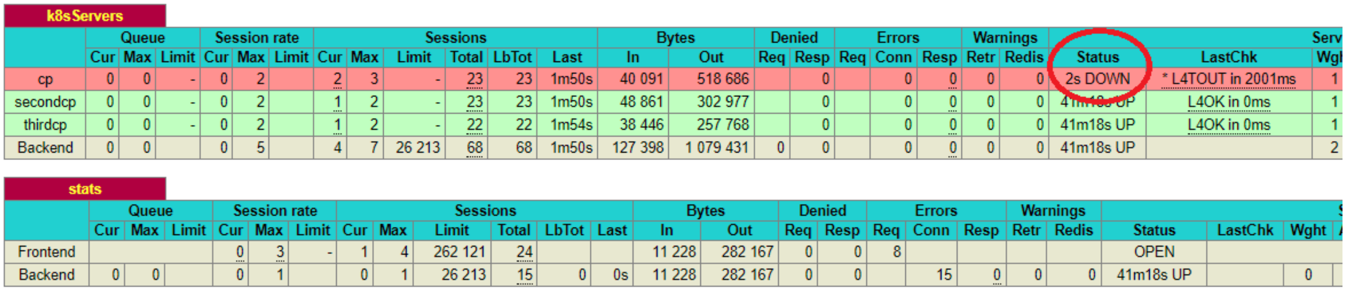
   ```
   student@cp:~$ sudo reboot
   ```

2. You will probably note the **logs** command exited when the service shut down. Run the same command and, among other output, you'll find errors similar to the following. Note the messages about losing the leader and electing a new one, with an eventual message that a peer has become inactive.

   ```
   student@cp:~$ kubectl -n kube-system logs -f etcd-thirdcp
   ```

   ```
   ....
   2023-08-09 02:11:39.569827 I | raft: 2331065cd4fb02ff [term: 9] received a MsgVote message with
   ↪  higher \
                                  term from ef44cc541c5f37c7 [term: 10]
   2023-08-09 02:11:39.570130 I | raft: 2331065cd4fb02ff became follower at term 10
   2023-08-09 02:11:39.570148 I | raft: 2331065cd4fb02ff [logterm: 9, index: 355240, vote: 0] cast
   ↪  MsgVote \
                                  for ef44cc541c5f37c7 [logterm: 9, index: 355240] at term 10
   2023-08-09 02:11:39.570155 I | raft: raft.node: 2331065cd4fb02ff lost leader d2620a7d27a9b449 at
   ↪  term 10
   2023-08-09 02:11:39.572242 I | raft: raft.node: 2331065cd4fb02ff elected leader ef44cc541c5f37c7
   ↪  at \
                                  term 10
   2023-08-09 02:11:39.682319 W | rafthttp: lost the TCP streaming connection with peer
   ↪  d2620a7d27a9b449 \
                                  (stream Message reader)
   2023-08-09 02:11:39.682635 W | rafthttp: lost the TCP streaming connection with peer
   ↪  d2620a7d27a9b449 \
                                  (stream MsgApp v2 reader)
   2023-08-09 02:11:39.706068 E | rafthttp: failed to dial d2620a7d27a9b449 on stream MsgApp v2 \
                                  (peer d2620a7d27a9b449 failed to find local node 2331065cd4fb02ff)
   2023-08-09 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 cp as down, but the other cp nodes remain up.



k8sServers

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings				Serv
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wgt
cp	0	0	-	0	2		2	3	-	23	23	1m50s	40 091	518 686		0		0	0	0	0	2s DOWN	* L4TOUT in 2001ms	1
secondcp	0	0	-	0	2		1	2	-	23	23	1m50s	48 861	302 977		0		0	0	0	0	41m18s UP	L4OK in 0ms	1
thirdcp	0	0	-	0	2		1	2	-	22	22	1m54s	38 446	257 768		0		0	0	0	0	41m18s UP	L4OK in 0ms	1
Backend	0	0		0	5		4	7	26 213	68	68	1m50s	127 398	1 079 431	0	0		0	0	0	0	41m18s UP		2

stats

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings				
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght
Frontend				0	3	-	1	4	262 121	24			11 228	282 167	0	0	8					OPEN		
Backend	0	0		0	1		0	1	26 213	15	0	0s	11 228	282 167	0	0		15	0	0	0	41m18s 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@secondcp:~$ kubectl -n kube-system exec -it etcd-secondcp -- /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
```

```
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 |  3.5.7  |   24 MB |      true |        12 |
↪ 395729 |
| 10.128.0.30:2379 | ef44cc541c5f37c7 |  3.5.7  |   24 MB |     false |        12 |
↪ 395729 |
+------------------+------------------+---------+---------+-----------+-----------+------------+
```

5. Turn the containerd service back on. You should see the peer become active and establish a connection.

```
student@cp:~$ sudo systemctl start containerd.service
```

```
student@cp:~$ kubectl -n kube-system logs -f etcd-thirdcp
```

```
....
2023-08-09 02:45:11.337669 I | rafthttp: peer d2620a7d27a9b449 became active
2023-08-09 02:45:11.337710 I | rafthttp: established a TCP streaming connection with peer\
                                d2620a7d27a9b449 (stream MsgApp v2 reader)
....
```

6. View the **etcd** cluster status again. Experiment with how long it takes for the **etcd** cluster to notice failure and choose a new leader with the time you have left.

Appendices

Appendix A

Domain Review

A.1 Exam Domain Review

Exercise A.1: Are you Ready?

1. If you have not searched for working and tested YAML examples, and can easily search for them again for all of the subjects the domain review mentions for the exam, you may:

   a. Run out of time while searching for good YAML examples.

   b. Make more mistakes.

   c. Use YAML that isn't proper for the Kubernetes version of the exam and waste time trying to troubleshoot the issue.

   d. All of the above.

2. Answer all that apply. In the context of the `Curriculum Overview` the term **Understand** when stating what a candidate should be able to do means:

   a. You only have a general idea what the object does.

   b. You can create the object.

   c. You can configure and integrate the object with other objects.

   d. You can properly update and test the object.

   e. You can troubleshoot the object.

3. Have you practiced creating, integrating, and troubleshooting all of the domain review items **at speed**?

   a. No. I kind of did the exercises. So I'm good.

   b. No. I did the labs twice over a two week period.

   c. Yes. I know the exam is intense and practiced with a clock running to make sure I can get everything done and also check my work.

Solution A.1

Are You Ready?

1. d.

2. b, c, d, e.
3. Hopefully c.

Exercise A.2: Preparing for the CKA Exam

Very Important

The source pages and content in this review could change at any time. **IT IS YOUR RESPONSIBILITY TO CHECK THE CURRENT INFORMATION**.

Before Taking Exam

Use this exercise as a resource after you complete the course but before you take the exam. Review the resources, know what good YAML looks like, and practice creating and working with objects at exam speed to assist with review and preparation.

1. Using a browser go to https://www.cncf.io/certification/cka/ and read through the program description.
2. In the **Exam Resources** section open the Curriculum Overview and Candidate-handbook in new tabs. Both of these should be read and understood prior to sitting for the exam.
3. Navigate to the Curriculum Overview tab. You should see links for domain information for various versions of the exam. Select the latest version, such as **CKA_Curriculum_V1.25.pdf**. The versions you see may be different. You should see a new page showing a PDF.
4. Read through the document. Be aware that the term `Understand`, such as `Understand Services`, is more than just knowing they exist. In this case expect it to also mean create, configure, update, and troubleshoot.
5. Using only the exam-allowed URLs and sub-domains search for YAML examples for each domain or skill item. Ensure it works for the version of the exam you are taking, as the YAML may not have been re-tested after a new release. Become familiar with out to find each good example again, so you can find the page again during the exam.
6. Using a timer see how long it takes you to create and verify the objects listed below. Write down the time. Try it again and see how much faster you can complete and test each step.

"Practice until you get it right. Then practice until you can't get it wrong" -Unknown

Domain Review Items

This list is copied from competency domains found on the PDF, and can be used as a checklist to ensure you have all the necessary YAML files and resources bookmarked. Again, **it remains your responsibility to check the web page for any changes to this list**.

- **Cluster Architecture, Installation & Configuration**
  - Manage role based access control (RBAC)
  - Use Kubeadm to install a basic cluster
  - Manage a highly-available Kubernetes cluster
  - Provision underlying infrastructure to deploy a Kubernetes cluster
  - Preform a version upgrade on a Kubernetes cluster using Kubeadm
  - Implement etcd backup and restore
- **Workloads & Scheduling**

  - Understand deployments and how to preform rolling updates and rollbacks
  - Use ConfigMaps and Secrets to configure applications
  - Know how to scale applications
  - Understand the primitives used to create robust, self-healing, application deployments
  - Understand how resource limits can affect Pod scheduling
  - Awareness of manifest management and common templating tools

- **Services & Networking**
  - Understand host networking configuration on the cluster nodes
  - Understand connectivity between Pods
  - Understand ClusterIP, NodePort, LoadBalancer service types and endpoints
  - Know how to use Ingress controllers and Ingress resources
  - Know how to configure and use CoreDNS
  - Choose an appropriate container network interface plugin

- **Storage**
  - Understand storage classes, persistent volumes
  - Understand volume mode, access modes and reclaim policies for volumes
  - Understand persistent volume claims primitive
  - Know how to configure applications with persistent storage

- **Troubleshooting**
  - Evaluate cluster and node logging
  - Understand how to monitor applications
  - Manage container stdout & stderr logs
  - Troubleshoot application failure
  - Troubleshoot cluster component failure
  - Troubleshoot networking

Exercise A.3: Practicing Skills

This exercise is to help you practice your skills. It does not cover all the items listed in the domain review guide. You should develop your own steps to build a full list of skill tests and steps.

Also note that all the detailed steps are not included. You should be able to complete these steps without being told what to type.

In a work or exam environment you may not be told exactly what to do or how to do it. The following steps are meant to get you used to thinking about solutions when the exact need isn't clear.

1. Find and use the `review1.yaml` file included in the course tarball. Use the **find** output and copy the YAML file to your home directory. Use **kubectl create** to create the object. Determine if the pod is running. Fix any errors you may encounter. The use of **kubectl describe** may be helpful.

```
student@cp:~$ find ~ -name review1.yaml

student@cp:~$ cp <copy-paste-from-above> .

student@cp:~$ kubectl create -f review1.yaml
```

2. After you get the pod running remove any pods or services you may have created as part of the review before moving on to the next section. For example:

```
student@cp:˜$ kubectl delete -f review1.yaml
```

3. Use the review2.yaml file to create a non-working deployment. Fix the deployment such that both containers are running and in a READY state. The web server listens on port 80, and the proxy listens on port 8080.

4. View the default page of the web server. When successful verify the GET activity logs in the container log. The message should look something like the following. Your time and IP may be different.

```
192.168.124.0 - - [3/Dec/2020:03:30:31 +0000] "GET / HTTP/1.1" 200 612 "-"
"curl/7.58.0" "-"
```

5. Find and use the review4.yaml file to create a pod, and verify it's running

6. Edit the pod such that it only runs on your worker node using the nodeSelector label.

7. Determine the CPU and memory resource requirements of design-pod1.

8. Edit the pod resource requirements such that the CPU limit is exactly twice the amount requested by the container. (Hint: subtract .22)

9. Increase the memory resource limit of the pod until the pod shows a Running status. This may require multiple edits and attempts. Determine the minimum amount necessary for the Running status to persist at least a minute.

10. Use the review5.yaml file to create several pods with various labels.

11. Using **only** the –selector value tux to delete only those pods. This should be half of the pods. Hint, you will need to view pod settings to determine the key value as well.

12. Create a new cronjob which runs busybox and the sleep 30 command. Have the cronjob run every three minutes. View the job status to check your work. Change the settings so the pod runs 10 minutes from the current time, every week. For example, if the current time was 2:14PM, I would configure the job to run at 2:24PM, every Monday.

13. Delete any objects created during this review. You may want to delete all but the cronjob if you'd like to see if it runs in 10 minutes. Then delete that object as well.

14. Create a new secret called specialofday using the key entree and the value meatloaf.

15. Create a new deployment called foodie running the nginx image.

16. Add the specialofday secret to pod mounted as a volume under the /food/ directory.

17. Execute a bash shell inside a foodie pod and verify the secret has been properly mounted.

18. Update the deployment to use the nginx:1.12.1-alpine image and verify the new image is in use.

19. Roll back the deployment and verify the typical, current stable version of nginx is in use again.

20. Create a new 200M NFS volume called reviewvol using the NFS server configured earlier in the lab.

21. Create a new PVC called reviewpvc which will uses the reviewvol volume.

22. Edit the deployment to use the PVC and mount the volume under /newvol

23. Execute a bash shell into the nginx container and verify the volume has been mounted.

24. Delete any resources created during this review.

25. Create a new deployment which uses the nginx image.

26. Create a new LoadBalancer service to expose the newly created deployment. Test that it works.

27. Create a new NetworkPolicy called netblock which blocks all traffic to pods in this deployment only. Test that all traffic is blocked to deployment.

28. Create a pod running nginx and ensure traffic can reach that deployment.

29. Update the `netblock` policy to allow traffic to the pod on port 80 only. Test that you can now access the default nginx web page.

30. Find and use the `review6.yaml` file to create a pod.

```
student@cp:~$ kubectl create -f review6.yaml
```

31. View the status of the pod.

32. Use the following commands to figure out why the pod has issues.

```
student@cp:~$ kubectl get pod securityreview
```

```
student@cp:~$ kubectl describe pod securityreview
```

```
student@cp:~$ kubectl logs securityreview
```

33. After finding the errors, log into the container and find the proper id of the nginx user.

34. Edit the pod such that the `securityContext` is in place and allows the web server to read the proper configuration files.

35. Create a new `serviceAccount` called `securityaccount`.

36. Create a ClusterRole named `secrole` which only allows create, delete, and list of pods in all apiGroups.

37. Bind the new clusterRole to the new serviceAccount.

38. Locate the token of the `securityaccount`. Create a file called `/tmp/securitytoken`. Put only the value of `token:` is equal to, a long string that may start with eyJh and be several lines long. Careful that only that string exists in the file.

39. Remove any resources you have added during this review

40. Create a new pod called `webone`, running the `nginx` service. Expose port 80.

41. Create a new service named `webone-svc`. The service should be accessible from outside the cluster.

42. Update both the pod and the service with selectors so that traffic for to the service IP shows the web server content.

43. Change the type of the service such that it is only accessible from within the cluster. Test that exterior access no longer works, but access from within the node works.

44. Deploy another pod, called `webtwo`, this time running the `wlniao/website` image. Create another service, called `webtwo-svc` such that only requests from within the cluster work. Note the default page for each server is distinct.

45. Test DNS names and verify CoreDNS is properly functioning.

46. Install and configure an ingress controller such that requests for webone.com see the `nginx` default page, and requests for webtwo.org see the `wlniao/website` default page. It does not matter which ingress controller you use.

47. Remove any resources created in this review.

48. Install a new cluster using an recent, previous version of Kubernetes. Backup etcd, then properly upgrade the entire cluster.

49. Create a pod running `busybox` without the scheduler being consulted.

50. Continue to create objects, integrate them with other objects and troubleshoot until each domain item has been covered.

Appendix B

Cilium Network Plugin

B.1 Labs

Exercise B.1: Install Cilium

Begin by installing the Cilium CLI onto your cp node. There are various architectures and releases, which can be found here https://github.com/cilium/cilium-cli/releases. Find and install the newest release for your architecture, such as amd64. Copying the URL from the website may be easier than typing out the URL. Also download the sha256sum file and compare it against the original.

```
wget  https://github.com/cilium/cilium-cli/releases/download/v0.12.4/cilium-linux-amd64.tar.gz{,.sha256sum}
```

```
<output_omitted>
```

```
sha256sum --check cilium-linux-amd64.tar.gz.sha256sum
```

```
cilium-linux-amd64.tar.gz: OK
```

Add the software and clean up the downloaded files

```
sudo tar xzvfC cilium-linux-amd64.tar.gz /usr/local/bin
rm cilium-linux-amd64.tar.gz{,.sha256sum}
```

Now use the cilium command to install and check the cluster:

```
cilium install
cilium status --wait
```

After installing the worker node, check connectivity:

```
cilium connectivity test
```

Explore other commands and features as time allows.