LFS458

Kubernetes Administration

Version 1.14.1

THE LINUX FOUNDATION

Published by:

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

If there are any questions about proper and fair use of the material herein, please contact:
training@linuxfoundation.org

Chapter 1

Introduction

1.1 Linux Foundation

What is the Linux Foundation?

- Nonprofit consortium dedicated to fostering the growth of **Linux**
- Promotes **Linux** and provides neutral collaboration and education
- Protects and supports **Linux** development
- Improves **Linux** as a technical platform
- Sponsors the work of **Linux** creator Linus Torvalds
- Supported by leading technology companies and developers

Linux Foundation Events

A partial list of **Linux Foundation** events. Some are held in multiple locations yearly, such as North America, Europe and Asia.

- Open Source Summit
- Linux Kernel Summit
- CloudOpen
- Open Source Leadership Summit
- Linux Security Summit
- Hyperledger Gloabl Forum
- KubeCon and CloudNativeCon
- Open Networking Summit
- LF Energy Summit
- OpenIot Summit
- OpenFinTech Summit
- Embedded Linux Conference
- Automotive Linux Summit
- KVM Forum
- Linux Storage Filesystem and MM Summit
- Cloud Foundry Summit

1.2 Linux Foundation Training

Training Venues

The **Linux Foundation** offers several types of training:

- Classroom
- Online
- On-Site
- Events Based

Training Offerings

Our current course offerings include:

- Linux Programming & Development Training
- Enterprise IT & Linux System Administration Courses
- Open Source Compliance Courses

For more information see `https://training.linuxfoundation.org`.

The **Linux Foundation** also offers a wide range of free **MOOC**s (**M**assively **O**pen **O**nline **C**ourses) offered through **edX** at `https://www.edx.org`. These cover basic as well as rather advanced topics associated with open source.

To find them at the **edX** website, search on `"Linux Foundation"`

Copyright

- The contents of this course and all its related materials, including handouts, are Copyright 2019 The Linux Foundation.
- No part of this publication may be stored in a retrieval system, transmitted or reproduced in any way, including, but not limited to, photocopy, photograph, magnetic, electronic or other record, without the prior written permission of The Linux Foundation.

1.3 Linux Foundation Certifications

LFCS and LFCE

The **Linux Foundation** offers a two-level **Certification Program**:

- **LFCS** (**L**inux **F**oundation **C**ertified **S**ysadmin)
- **LFCE** (**L**inux **F**oundation **C**ertified **E**ngineer)

Full details about this program can be found at `https://training.linuxfoundation.org/certification`.

This information includes a thorough description of the **Domains** and **Competencies** covered by each exam.

Certification/Training Firewall

- The **Linux Foundation** has two separate training divisions:
  - Certification
  - Course Delivery
- These are separated by a **firewall**:
  - Enables third party organizations to develop and deliver **LF** certification preparation classes
  - Prevents using ***secret sauce*** in **LF** courses
  - Prevents ***teaching the test*** in **LF** courses
- Instructors (including today) are guided entirely by publicly available information

Preparation Resources

- Before doing anything else, download:
  https://training.linuxfoundation.org/download-free-certification-prep-guide
- Sections on:
  - Domains and Competencies
  - Free Training Resources
  - Paid Training Resources
  - Taking the Exam
- Also get the **Candidate Handbook** at:

https://training.linuxfoundation.org/go/candidate_handbook

- Also read exam-specific information at:

https://training.linuxfoundation.org/certification/lfcs

https://training.linuxfoundation.org/certification/lfce

1.4 Laboratory Exercises, Solutions and Resources

Labs

- Hands-on exercises provided at the end of each session.
- Can be done on either virtual machines or bare-metal
  - Unless otherwise specified
- Some exercises marked as optional
- Solutions available at the end of each exercise.

Obtaining Course Solutions and Resources

- If this course has such material, lab exercise solutions, suggestions and other resources may be downloaded from: https://training.linuxfoundation.org/cm/LFS458
- If you do not have a browser available you can obtain with:

```
$ wget --user=LFtraining --password=<ask instructor> \
  https://training.linuxfoundation.org/cm/LFS458/LFS458_V1.14.1_SOLUTIONS.tar.bz2
```

  and similary for the RESOURCES file.

```
$ wget --user=LFtraining --password=<ask instructor> \
  https://training.linuxfoundation.org/cm/LFS458/LFS458_V1.14.1_RESOURCES.tar
```

- Any errata, updated solutions, etc. will also be posted on that site
- These files may be unpacked with:

```
$ tar xvf LFS458_*SOLUTIONS*.tar.bz2
$ tar xvf LFS458_*RESOURCES*.tar
```

1.5 Distribution Details

Software Environment

- Class material is designed for multiple environments
- Focuses on three current major **Linux** distribution families:
  - **Red Hat / Fedora**
  - **OpenSUSE / SUSE**
  - **Debian**

Which Distribution to Choose

- Several factors to consider:
  - Has your employer already standardized?
  - Do you want to learn more?
  - Do you want to certify?
- You are encouraged to experiment with more than one distribution

Red Hat / Fedora Family

- Current material based upon the latest release of **Red Hat Enterprise Linux (RHEL)** 7.x at the time of publication, and should work well with later versions.
- Supports **x86**, **x86-64**, **Itanium**, **PowerPC** and **IBM System Z**
- RPM-based, uses **yum** (or **dnf**) to install and update
- Long release cycle; targets enterprise server environments
- Upstream for **CentOS**, **Scientific Linux** and **Oracle Linux**

> **Please Note: CentOS** is used for demos and labs because it is available at no cost.

OpenSUSE Family

- Current material based upon the latest release of **OpenSUSE** and should work well with later versions.
- RPM-based, uses **zypper** to install and update.
- **YaST** available for administration purposes
- **x86** and **x86-64**
- Upstream for **SUSE Linux Enterprise Server (SLES)**

> **Please Note: OpenSUSE** is used for demos and labs because it is available at no cost.

Debian Family

- Commonly used on both servers and desktop
- **DPKG**-based, use **apt** and front-ends for installing and updating
- Upstream for **Ubuntu**, **Linux Mint** and others
- Current material based upon the latest release of **Ubuntu** and should work well with later versions.
- **x86** and **x86-64**
  - Long Term Release (LTS)

Please Note: Ubuntu is used for demos and labs because it is available at no cost, as is **Debian**, but has a wider base with new **Linux** users.

New Distribution Similarities

- Current trends and changes to the distributions have reduced some of the differences between the distributions.
  - **systemd**, system startup and service management
  - **journald**, manage system logs
  - **firewalld**, firewall management daemon
  - **ip**, network display and configuration tool

Please Note: Since these utilities are common across distributions, the lecture and lab information will be mostly based on them.

If your choice of distribution or release does not support these commands, please translate accordingly.

AWS Free Tier

- **Amazon Web Services** (**AWS**) offers a wide range of virtual machine products (**instances**) which remote users can access in the cloud.
- In particular, one can use their **Free Tier** account level for up to a year and the simulated hardware and software choices available may be all one needs to perform the exercises for **Linux Foundation** training courses and gain experience with **Linux**. Or they may furnish a very educational supplement to working on local hardware, and offer opportunities to easily study more than one **Linux** distribution.
- For kernel-level courses there are some particular challenges. Please download a helpful guide we have prepared to help you experiment with the **AWS** free tier: `https://training.linuxfoundation.org/cm/prep/aws.pdf`.

1.6 Labs

Exercise 1.1: Configuring the System for sudo

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

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

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

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

```
$ sudo ls
```

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

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

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

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

if the user is `student`.

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

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

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


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

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

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

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

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

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

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

Chapter 2

Basics of Kubernetes

2.1 Define Kubernetes

What Is Kubernetes

- Orchestration
- `http://kubernetes.io`
  "Open-source software for automating deployment, scaling, and management of containerized applications"
- Easy to run, Potentially complex to integrate
- Built with lessons from **Google**
- Open and extensible
- From Greek κνβερνητης, pilot or Helmsman

2.2 Cluster Structure

Components of Kubernetes

- microservices
- Decoupling
- Built to be transient
- API call driven
- YAML to JSON, written in **Go**

Figure 2.1: **K8s Official Logo**

2.3 Adoption

Challenges

- Easy to build container images
- Simple to share images via registries
- Powerful user tools to manage containers
- Network considerations
- Flexible storage

Other Solutions

- Docker Swarm
- Apache Mesos
- Nomad
- Rancher

Figure 2.2: **Official Project Logos**

Borg Heritage

- Orchestration system to manage all **Google** applications at scale
- Described publicly in 2015

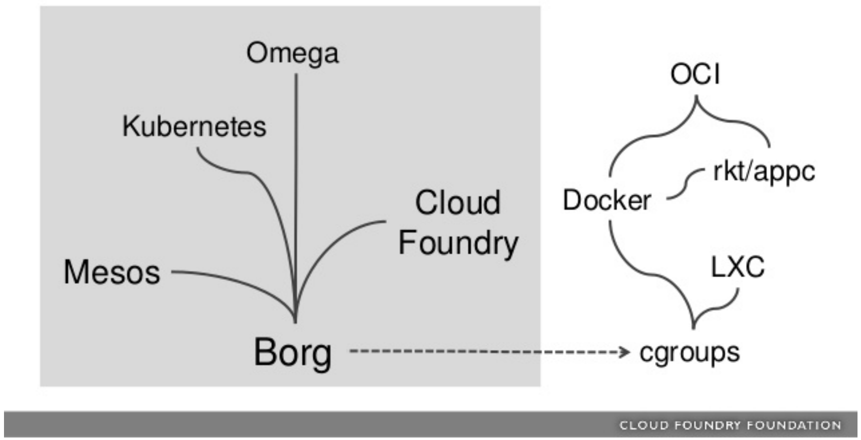

Figure 2.3: **The Kubernetes Lineage (by Chip Childers, Cloud Foundry Foundation, from LinkedIn Slideshare)**

Kubernetes Architecture

- Central manager
  - API server, Scheduler, Controllers, and etcd storage system
- Worker nodes

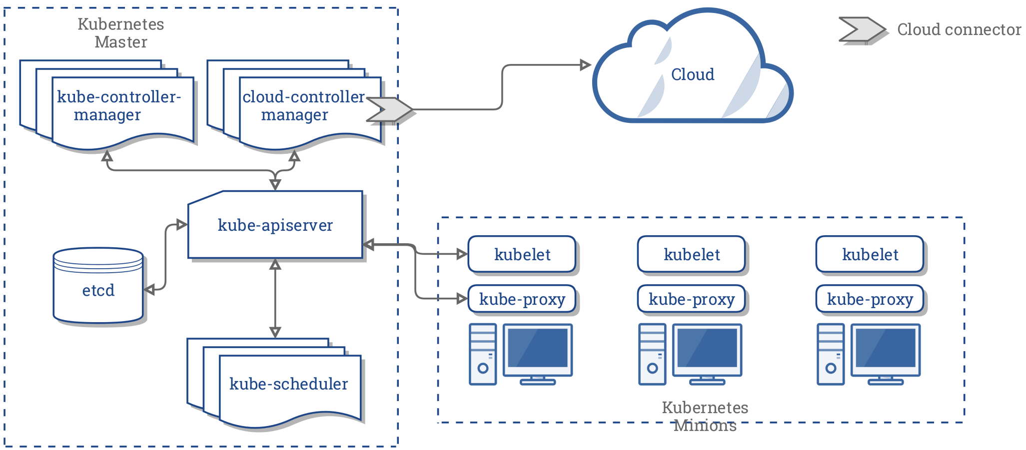

Figure 2.4: **Kubernetes Architecture**

Terminology

- Pod
- Controller
  - ReplicaSet
  - Deployment
  - DaemonSet
  - Jobs
- Label
- Taint
- Toleration
- Annotation

Innovation

- Given to open source June 2014
- Thousands of contributors
- More than 73K commits
- More than 23K on **Slack**
- Currently three month major release cycle
- Constant change

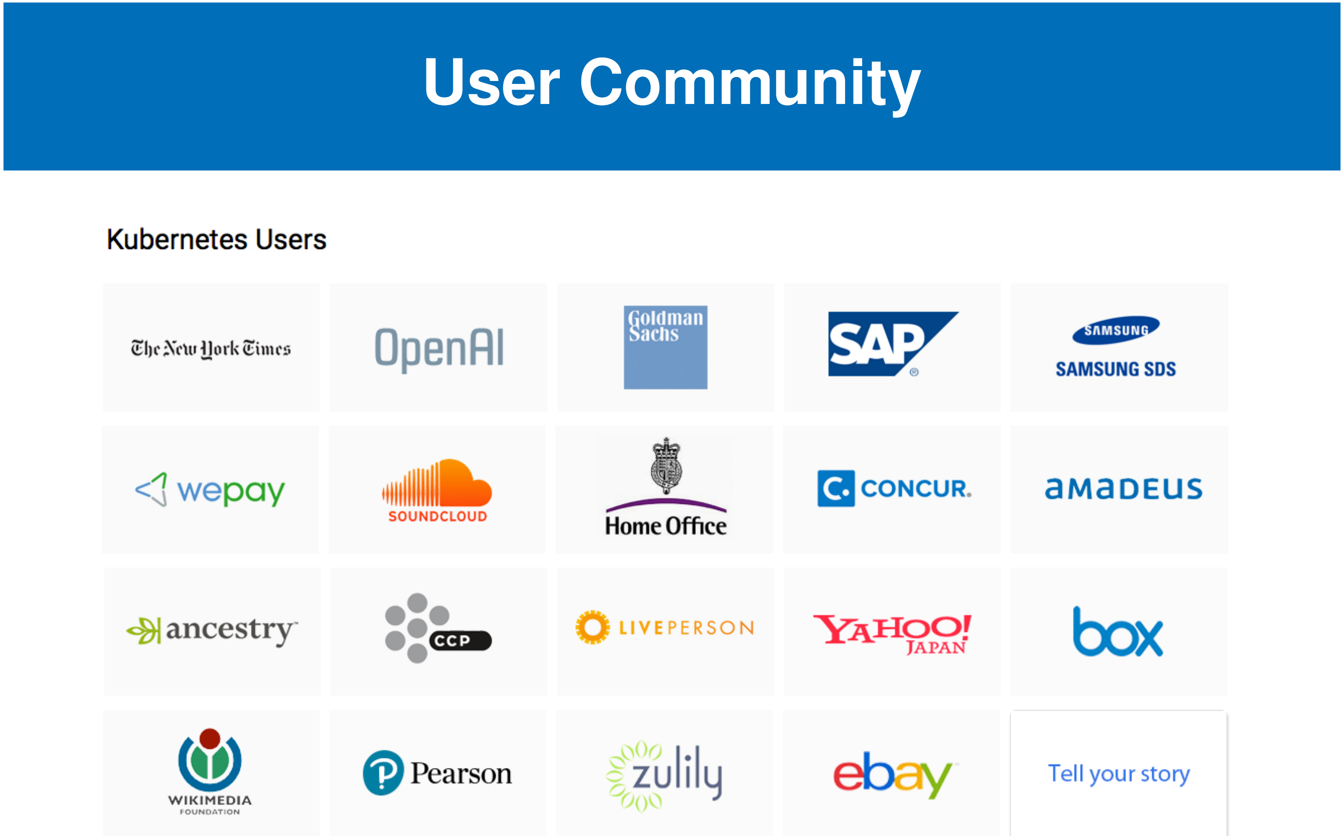

Figure 2.5: **Kubernetes Users**

Tools

- Several tools to choose from
  - Minikube
  - kubeadm
  - kubectl
  - Dashboard
  - kubefed
  - kops
  - helm
  - kompose

2.4 Project Governance and CNCF

Cloud Native Computing Foundation

Figure 2.6: **CNCF**

Resource Recommendations

- The Borg Paper
- John Wilkes speech
- Local community hangout
- Slack channel
- Stack Overflow community

2.5 Labs

Exercise 2.1: View Online Resources

Visit kubernetes.io

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

1. Open a browser and visit the `https://kubernetes.io/` website.
2. In the upper right hand corner, use the drop down to view the versions available. It will say something like `v1.12`.
3. Select the top level link for `Documentation`. The links on the left of the page can be helpful in navigation.
4. As time permits navigate around other sub-pages such as `SETUP`, `CONCEPTS`, and `TASKS` to become familiar with the layout.

Track Kubernetes Issues

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

1. To view the main page use your browser to visit `https://github.com/kubernetes/kubernetes/`
2. Click on various sub-directories and view the basic information available.
3. Update your URL to point to `https://github.com/kubernetes/kubernetes/issues`. You should see a series of issues, feature requests, and support communication.

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

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

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

7. Explore the page with the remaining time left.

Chapter 3

Installation and Configuration

3.1 Getting Started With Kubernetes

Installation Tools

- Google Kubernetes Engine (**GKE**)
- **Minikube**
- **Kubespray**
- Kubernetes Operations (**kops**)
- Both need **kubectl**
- **kubeadm** newer tool

Installing kubectl

- Install or compile **kubectl**
- Main binary for working with objects
- Available for common distributions via dedicated repos
- Configuration file: `~/.kube/config`
  - endpoints
  - SSL keys
  - contexts

Using Google Kubernetes Engine (GKE)

- Create account on **GKE**
- Add method of payment
- Install and use **gcloud**
  - Vendor-specific command to manage GKE
- More details:
  https://console.cloud.google.com/getting-started

3.2 Minikube

Using Minikube

- Open source project within GitHub Kubernetes
- Download from **Google**
- Assumes **VirtualBox** already installed
- Useful for developers
- Uses **Go** binary **localkube**
- Also uses **Docker**

3.3 kubeadm

Install With kubeadm

- Available since Kubernetes 1.4.0
- Works with **Ubuntu 16.04** and **CentOS 7.1**
- Main steps
  - Run `kubeadm init` on the head node
  - Create a network for IP-per-Pod criteria
  - Run `kubeadm join --token` *`token head-node-IP`* on worker nodes

Install A Pod Network

- Only one pod network per cluster
- Several to choose from
  - **Calico**
  - **Canal**
  - **Flannel**
  - **Kube-router**
  - **Romana**
  - **Weave Net**
- Can become complicated to manage
- Several add-ons available

3.4 More Installation Tools

More Installation Tools

- **kubespray**
- **kops**
- **kube-aws**
- **kubicorn**

Installation Considerations

- Which provider should I use?
  - Public or private cloud?
- Which operating system should I use?
- Which networking solution should I use?
  - Do I need an overlay?
- Where should I run my **etcd** cluster?
- Should I configure Highly-Available head nodes?

Main Deployment Configurations

- Single-node
- Single head node, multiple workers
- Multiple head nodes with HA, multiple workers
- HA etcd, HA head nodes, multiple workers
- Federation also provides higher availability

systemd Unit File for Kubernetes

```
  - name: kube-controller-manager.service
      command: start
      content: |
        [Unit]
        Description=Kubernetes Controller Manager
Documentation=https://github.com/kubernetes/kubernetes
        Requires=kube-apiserver.service
        After=kube-apiserver.service
        [Service]
        ExecStartPre=/usr/bin/curl -L -o /opt/bin/kube-controller-manager
          -z /opt/bin/kube-controller-manager \
          https://storage.googleapis.com/kubernetes-release/release/v1.4.0/\
          bin/linux/amd64/kube-controller-manager
        ExecStartPre=/usr/bin/chmod +x /opt/bin/kube-controller-manager
        ExecStart=/opt/bin/kube-controller-manager \
          --service-account-private-key-file=/opt/bin/kube-serviceaccount.key \
          --root-ca-file=/var/run/kubernetes/apiserver.crt \
          --master=127.0.0.1:8080 \
....
```


Using Hyperkube

```
$ docker run --rm gcr.io/google_containers/hyperkube:v1.10.12 \
                  /hyperkube apiserver --help

$ docker run --rm gcr.io/google_containers/hyperkube:v1.10.12 \
                  /hyperkube scheduler --help

$ docker run --rm gcr.io/google_containers/hyperkube:v1.10.12 \
                  /hyperkube controller-manager --help
```

Compiling from Source

- Configure **Golang** environment
- Clone source code
- May need to install other compiler and libraries

3.5 Labs

Exercise 3.1: Install Kubernetes

Overview

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

Platforms: GCP, AWS, VirtualBox, etc

The labs were written using **Ubuntu** instances running on **G**oogle **C**loud **P**latform (**GCP**). They have been written to be vendor-agnostic so could run on AWS, local hardware, or inside of virtualization to give you the most flexibility and options. Each platform will have different access methods and considerations. As of v1.12.1 the minimum (as in barely works) size for **VirtualBox** is 3vCPU/1G memory/5G minimal OS for `master` and 1vCPU/1G memory/5G minimal OS for worker node.

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

Very Important

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

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

YAML files and White Space

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

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

`https://training.linuxfoundation.org/cm/LFS458`

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

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

```
$ wget https://training.linuxfoundation.org/cm/LFS458/LFS458_V1.14.1_SOLUTIONS.tar.bz2 \
            --user=LFtraining --password=Penguin2014

$ tar -xvf LFS458_V1.14.1_SOLUTIONS.tar.bz2
```


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

Bionic

While **Ubuntu 18** `bionic` has become the typical version to deploy, the Kubernetes repository does not yet have compatible binaries at the time of this writing. While `xenial` binaries can be used there are many additional steps necessary to complete the labs. A **Ubuntu 18** version is expected to be available soon.

Install Kubernetes

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

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

```
[student@laptop ~]$  ssh -i LFS458.pem student@35.226.100.87
The authenticity of host '54.214.214.156 (35.226.100.87)' can't be established.
ECDSA key fingerprint is SHA256:IPvznbkx93/Wc+ACwXrCcDDgvBwmvEXC9vmYhk2Wo1E.
ECDSA key fingerprint is MD5:d8:c9:4b:b0:b0:82:d3:95:08:08:4a:74:1b:f6:e1:9f.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '35.226.100.87' (ECDSA) to the list of known hosts.
<output_omitted>
```

2. Become `root` and update and upgrade the system. Answer any questions to use the defaults.


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

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

<output_omitted>
```

3. The main choices for a container environment are **Docker** and **cri-o**. We will user **Docker** for class, as **cri-o** requires a fair amount of extra work to enable for Kubernetes. As **cri-o** is open source the community seems to be heading towards its use.

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

<output-omitted>
```

4. Add new repo for kubernetes. You could also get a tar file or use code from GitHub. Create the file and add an entry for the main repo for your distribution. As we are still using `Ubuntu 16.04` add the `kubernetes-xenial` with the key word `main`. Note there are four sections to the entry.

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

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

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

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

OK
```

6. Update with new repo, which will download new repo information.

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

<output-omitted>
```


7. Install the software. There are regular releases the newest of which can be used by omitting the equal sign and version information on the command line. Historically new version have lots of changes and a good chance of a bug or five.

```
root@lfs458-node-1a0a:~# apt-get install -y \
        kubeadm=1.14.1-00 kubelet=1.14.1-00 kubectl=1.14.1-00

<output-omitted>
```

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

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

   We will use **Calico** as a network plugin which will allow us to use `Network Policies` later in the course. Currently **Calico** does not deploy using CNI by default. The 3.3 version of **Calico** has more than one configuration file for flexibility with RBAC. Download the configuration files for. Once downloaded look for the expected IPV4 range for containers to use.

   A short url for each file is shown in the following **wget** commands. the longer URLs can be found here: https://docs.projectcalico.org/v3.3/getting-started/kubernetes/installation/hosted/rbac-kdd.yaml and: https://docs.projectcalico.org/v3.3/getting-started/kubernetes/installation/hosted/kubernetes-datastore/calico-networking/1.7/calico.yaml

```
root@lfs458-node-1a0a:~# wget https://tinyurl.com/yb4xturm -O rbac-kdd.yaml
root@lfs459-node-1a0a:~# wget https://tinyurl.com/y8lvqc9g -O calico.yaml
```

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

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

calico.yaml

```
....
          # Configure the IP Pool from which Pod IPs will be chosen.
           - name: CALICO_IPV4POOL_CIDR
             value: "192.168.0.0/16"
....
```

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

```
root@lfs458-node-1a0a:~# kubeadm init \    # This creates the cluster
     --kubernetes-version 1.14.1 \         # To avoid using newest possible
     --pod-network-cidr 192.168.0.0/16 \    # The IP range we found in Calico file
      | tee kubeadm-init.out      # Save output for future review
```

Please Note

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

```
[init] using Kubernetes version: v1.14.1
[preflight] running pre-flight checks
[preflight/images] Pulling images required for setting up a
  Kubernetes cluster
[preflight/images] This might take a minute or two, depending
  on the speed of your internet connection

<output-omitted>
```



```
Your Kubernetes master has initialized successfully!

To start using your cluster, you need to run the following as
 a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options
listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

You can now join any number of machines by running the following
on each node as root:

  kubeadm join 10.128.0.4:6443 --token rdnhok.g8mb6lfgesunanvh
--discovery-token-ca-cert-hash
sha256:66350d154fc0169b5bb5fd50c04b72468195e356d78d95f137ed55e995402f77
```

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

```
root@lfs458-node-1a0a:~# exit
logout

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

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

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

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

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

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

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

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

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

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

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

```
configmap/calico-config created
service/calico-typha created
deployment.apps/calico-typha created
poddisruptionbudget.policy/calico-typha created
<output_omitted>
```

13. While many objects have short names, a **kubectl** command can be a lot to type. We will enable **bash** auto-completion. Begin by adding the settings to the current shell. Then update the `/.bashrc` file to make it persistent.

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

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

14. 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, then look at pods in the `kube-system` namespace.

```
student@lfs458-node-1a0a:~$ kubectl des<Tab> n<Tab><Tab> lfs458-<Tab>

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

Exercise 3.2: Grow the Cluster

Open another terminal and connect into a your second node. Install **Docker** and Kubernetes software. These are the many, but not all, of the steps we did on the master node.

This book will use the **lfs458-worker** prompt for the node being added to help keep track of the proper node for each command. Note that the prompt indicates both the user and system upon which run the command.

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

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

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

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

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

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

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

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

```
root@lfs458-worker:~# apt-get install -y \
      kubeadm=1.14.1-00 kubelet=1.14.1-00 kubectl=1.14.1-00

root@lfs458-worker:~# exit
```

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

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

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

3. At this point we could copy and paste the **join** command from the master node. That command only works for 24 hours, so we will build our own **join** should we want to add compute nodes in the future. Find the token on the master node. The token lasts 24 hours by default. If it has been longer, and no token is present you can generate a new one with the **sudo kubeadm token create** command, seen in the following command.

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

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

4. **Only if the token has expired**, you can create a new token, to use as part of the join command.

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

27eee4.6e66ff60318da929
```

5. Starting in v1.9 you should create and use a Discovery Token CA Cert Hash created from the master to ensure the node joins the cluster in a secure manner. Run this on the master node or wherever you have a copy of the CA file. You will get a long string as output.

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



```
6d541678b05652e1fa5d43908e75e67376e994c3483d6683f2a18673e5d2a1b0
```

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

```
root@lfs458-worker:~# kubeadm join \
     --token 27eee4.6e66ff60318da929 \
     10.128.0.3:6443 \
     --discovery-token-ca-cert-hash \
     sha256:6d541678b05652e1fa5d43908e75e67376e994c3483d6683f2a18673e5d2a1b0

[preflight] Running pre-flight checks.
        [WARNING FileExisting-crictl]: crictl not found in system path
[discovery] Trying to connect to API Server "10.142.0.2:6443"
[discovery] Created cluster-info discovery client, requesting info from
"https://10.142.0.2:6443"
[discovery] Requesting info from "https://10.142.0.2:6443" again to
validate TLS against the pinned public key
[discovery] Cluster info signature and contents are valid and TLS
certificate validates against pinned roots, will
use API Server "10.142.0.2:6443"
[discovery] Successfully established connection with API Server
"10.142.0.2:6443"
This node has joined the cluster:
* Certificate signing request was sent to master and a response
  was received.
* The Kubelet was informed of the new secure connection details.
Run 'kubectl get nodes' on the master to see this node join the cluster.
```

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

```
root@lfs458-worker:~# exit

student@lfs458-worker:~$ kubectl get nodes
```

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

```
student@lfs458-worker:~$ ls -l .kube
```

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

Exercise 3.3: Finish Cluster Setup

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

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

```
NAME               STATUS   ROLES    AGE   VERSION
lfs458-node-1a0a   Ready    master   28m   v1.14.1
lfs458-worker      Ready    <none>   50s   v1.14.1
```

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

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

```
Name:               lfs458-node-1a0a
Roles:              master
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/hostname=lfs458-node-1a0a
                    node-role.kubernetes.io/master=
Annotations:        kubeadm.alpha.kubernetes.io/cri-socket: /var/run/dockershim.sock
                    node.alpha.kubernetes.io/ttl: 0
                    projectcalico.org/IPv4Address: 10.142.0.3/32
```



```
                    volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp:  Mon, 07 Jan 2019 22:04:03 +0000
Taints:             node-role.kubernetes.io/master:NoSchedule
<output_omitted>
```

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

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

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


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


node/lfs458-node-1a0a untainted
error: taint "node-role.kubernetes.io/master:" not found
```

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

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

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


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


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

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

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

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

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

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

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

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

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

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

```
student@lfs458-node-1a0a:~$ ip a
```

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

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 an Kubernetes object while will deploy and monitor an application in a container. Verify it is running and the desired number of container matches the available.

```
student@lfs458-node-1a0a:~$ kubectl create deployment nginx --image=nginx

deployment.apps/nginx created


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

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

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

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

Name:                   nginx
Namespace:              default
CreationTimestamp:      Mon, 23 Apr 2019 22:38:32 +0000
```

```
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=nginx
Replicas:               1 desired | 1 updated | 1 total | 1 ava....
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
<output_omitted>
```

3. View the basic steps the cluster took in order to pull and deploy the new application. You should see several lines of output with newer events at the top.

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

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  creationTimestamp: 2017-09-27T18:21:25Z
<output_omitted>
```

5. Run the command again and redirect the output to a file. Then edit the file. Remove the `creationTimestamp`, `resourceVersion`, `selfLink`, and `uid` lines. Also remove all the lines including and after `status:`, which should be somewhere around line 40, if others have already been removed.

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

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



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

6. Delete the existing deployment.

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

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

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

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

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

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

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

```
student@lfs458-node-1a0a:~$ kubectl create deployment two --image=nginx --dry-run -o yaml
```

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

```
    run: two
  name: two
spec:
<output_omitted>
```

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

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

10. Existing objects can be viewed in a ready to use YAML output. Take a look at the existing **nginx** deployment. Note there is more detail to the **–export** option. The flag has been **deprecated** and may be removed in the future.

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


```
apiVersion: extensions/v1beta1
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@lfs458-node-1a0a:~$ kubectl get deployment nginx --export -o json
```


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

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

```
student@lfs458-node-1a0a:~$ kubectl expose deployment/nginx
error: couldn't find port via --port flag or introspection
See 'kubectl expose -h' for help and examples.
```

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

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

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

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

first.yaml

```
....
  spec:
    containers:
```



```yaml
        - 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@lfs458-node-1a0a:~$ kubectl replace -f first.yaml
deployment.extensions/nginx replaced
```

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

```
student@lfs458-node-1a0a:~$ kubectl get deploy,pod
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/nginx    1/1     1            1           2m4s

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

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

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

18. Verify the service configuration. First look at the service information, then at the endpoint information. Note the Cluster IP is not the current endpoint. Calico is Cluster IP to the Endpoint handled by `kubelet` and `kube-proxy`. Take note of the current endpoint IP. In the example below it is `192.168.1.580`:. We will use this information in a few steps.


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

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

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

NAME    ENDPOINTS        AGE
nginx   192.168.1.5:80   26s
```

19. Determine which node the container is running on. Log into that node and use **tcpdump** to view traffic on the `tunl0`, as in tunnel zero, interface. The second node in this example. You may also see traffic on an interface which starts with `cali` and some string. Leave that command running while you run **curl** in the following step. You should see several messages go back and forth, including a `HTTP` HTTP/1.1 200 OK: and a `ack` response to the same sequence.

```
student@lfs458-node-1a0a:~$ kubectl describe pod nginx-7cbc4b4d9c-d27xw \
                  | grep Node:

Node:    lfs458-worker/10.128.0.5
```

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

tcpdump: verbose output suppressed, use -v or -vv for full protocol...
listening on tunl0, link-type EN10MB (Ethernet), capture size...
<output_omitted>
```

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

```
student@lfs458-node-1a0a:~$ curl 10.100.61.122:80

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

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

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

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

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

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

deployment.extensions/nginx scaled
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
student@lfs458-node-1a0a:~$ curl 10.100.61.122:80

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

Exercise 3.5: Access from Outside the Cluster

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

1. Begin by getting a list of the pods.

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

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

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

```
student@lfs458-node-1a0a:~$ kubectl exec nginx-1423793266-13p69 \
    -- printenv |grep KUBERNETES

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

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

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

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

4. Delete the service.

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

service "nginx" deleted
```

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

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

service/nginx exposed

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

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

6. Open a browser on your local system, not the GCE node, and use the public IP of your node and port `32753`, shown in the output above. If running the labs on a remote system like **AWS** or **GCE** the CLUSTER-IPs are internal. Use the public IP you used with SSH to gain access.

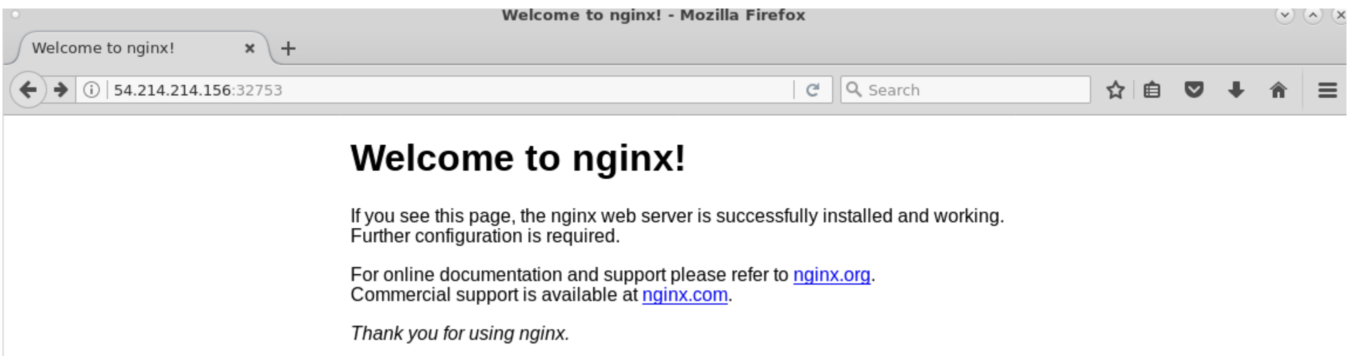

Figure 3.1: **External Access via Browser**

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


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

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

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

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

student@lfs458-node-1a0a:~$ kubectl get po
NAME                      READY     STATUS    RESTARTS   AGE
nginx-1423793266-7x181    1/1       Running   0          6s
nginx-1423793266-s6vcz    1/1       Running   0          6s
```

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

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

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

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

Chapter 4

Kubernetes Architecture

4.1 Kubernetes Architecture

Main Components

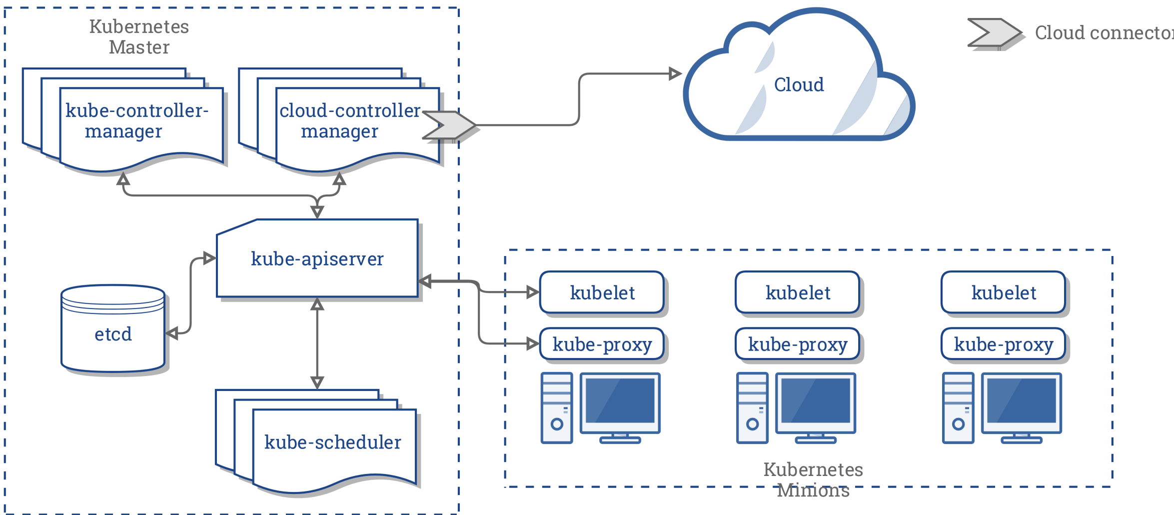

Figure 4.1: **K8s Architectural Overview from kubernetes.io**

Master Node

- **kube-apiserver**
- **kube-scheduler**
- **etcd**
- **kube-controller-manager**
- **cloud-controller-manager**
- Add-ons
  – DNS
  – Dashboard - Web UI
  – Cluster-level resource monitoring
  – cluster-level logging

kube-apiserver

- Front-end of cluster’s shared state
- Master for the cluster
- All components work through it
- Validates and configures data for API objects
- Services REST operations
- Only component to connect to **etcd** database

kube-scheduler

- Uses algorithm to determine Pod placement
- Checks quota restrictions
- Custom scheduling policies possible
- Affinity rules to place pods on specific nodes
- `Taints` can be used to repel pods
- Pod bindings can force particular scheduling

etcd Database

- Multiversion persistent b+tree key-value store
- Append only, regular compaction
- Shed oldest version of superseded data
- Works with **curl** and other HTTP libraries
- Provides reliable watch queries
- Distributed consensus protocol for leadership

Other Agents

- **kube-controller-manager**
  - Daemon which embeds core control-loops
  - Watches state of cluster
  - Works to make current state match desired state
- **cloud-controller-manager (ccm)**
  - Interacts with outside cloud managers
  - Allows features to be developed outside of core release cycle
  - Each **kubelet** must use `--cloud-provider-external`
  - Handles tasks once part of `kube-controller-manager`

Worker Nodes

- kubelet
- kube-proxy
- Docker engine or rkt
- Supervisord
- Fluentd

kubelet

- Agent on each node
- Uses `PodSpec`
- Mounts volumes to Pod
- Downloads `secrets`
- Passes request to local container engine
- Reports status of Pods and node to cluster

Services

- Connect Pods together
- Expose Pods to Internet
- Decouple settings
- Define Pod access policy
- Microservice

Controllers

- Watch based control loop monitoring delta
- `Informer / SharedInformer`
- `Workqueue`
- Shipped controllers
  - `replication` controller
  - `endpoints` controller
  - `namespace` controller
  - `serviceaccounts` controller

Pods

- One or more containers
- Smallest unit to work with
- Only one, shared IP address per Pod

Containers

- Not worked with directly
- Usage limits passed to container engine

```
resources:
  limits:
    cpu: "1"
    memory: "4Gi"
  requests:
    cpu: "0.5"
    memory: "500Mi"
```

- ResourceQuota
- PriorityClass

Component Review

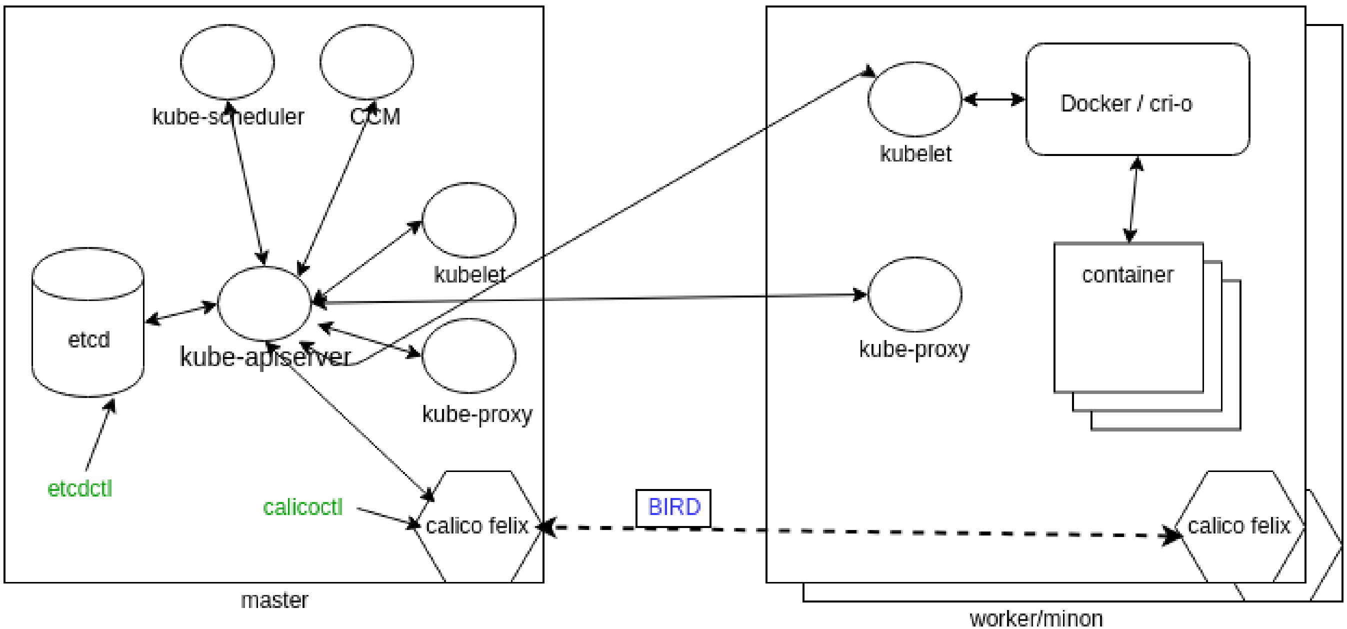

Figure 4.2: **K8s Architectural Review**

Node

- Created outside of cluster
- **NodeStatus**
- **NodeLease**

4.2 Networking

Single IP per Pod

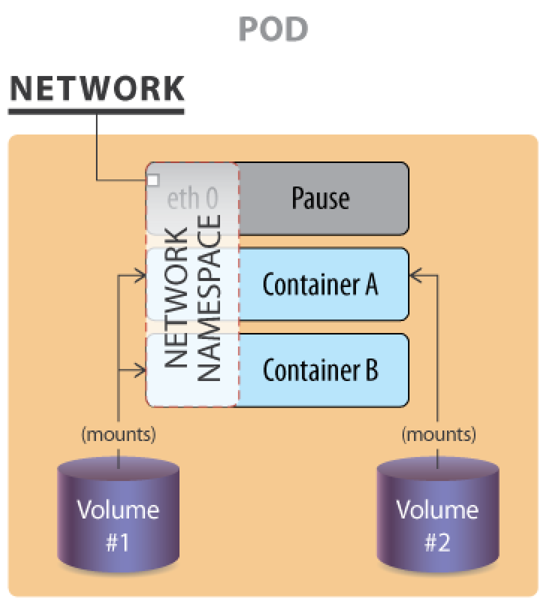

Figure 4.3: **Pod Network**

Container to Outside Path

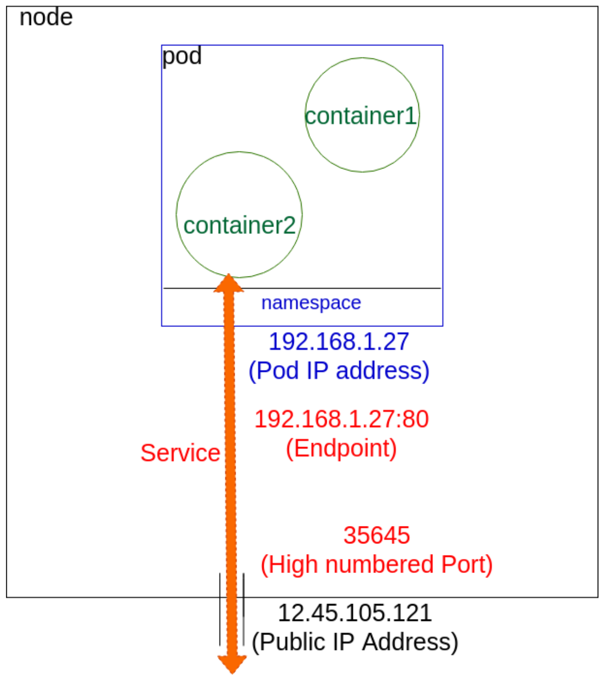

Figure 4.4: **Container Network**

Networking Setup

- Main networking challenges:
  - Coupled container-to-container communications
  - Pod-to-pod communications
  - External-to-pod communications (solved by the services concept, to be discussed later)
- Admin configuration required
- IP assigned to pod, not container

CNI Network Configuration File

```
{
    "cniVersion": "0.2.0",
    "name": "mynet",
    "type": "bridge",
    "bridge": "cni0",
    "isGateway": true,
    "ipMasq": true,
    "ipam": {
        "type": "host-local",
        "subnet": "10.22.0.0/16",
        "routes": [
            { "dst": "0.0.0.0/0" }
             ]
    }
}
```

Pod-to-Pod Communication

- All pods can communicate with each other across nodes.
- All nodes can communicate with all pods.
- No Network Address Translation (NAT).

4.3 Other Cluster Systems

Mesos

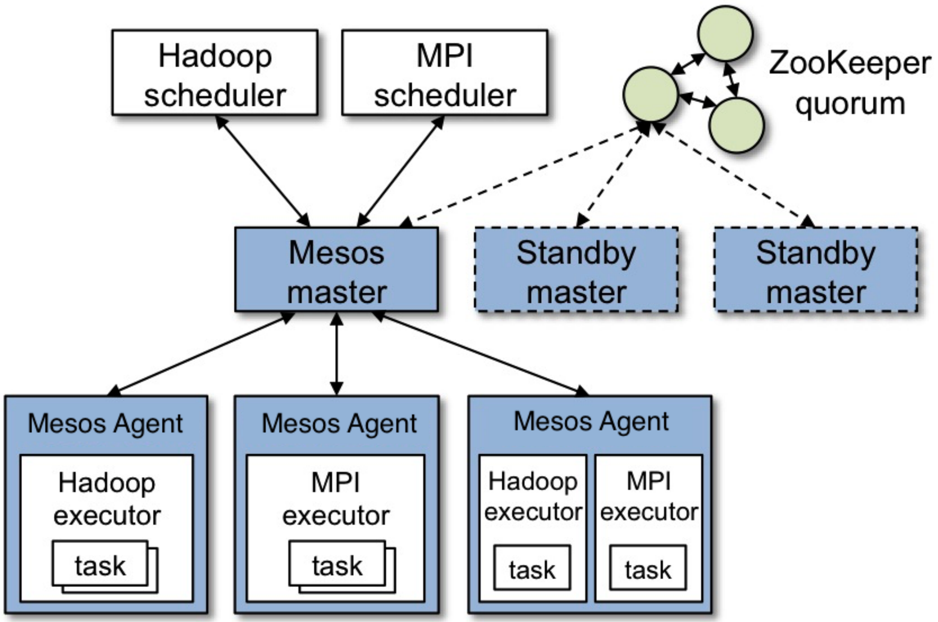

Figure 4.5: **Mesos Architecture**

4.4 Labs

Exercise 4.1: Working with CPU and Memory Constraints

Overview

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

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

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

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

student@lfs458-node-1a0a:~$ kubectl get deployments
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
hog     1/1     1            1           13s
```

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

```
student@lfs458-node-1a0a:~$ kubectl describe deployment hog
Name:                   hog
Namespace:              default
CreationTimestamp:      Tue, 08 Jan 2019 17:01:54 +0000
Labels:                 app=hog
Annotations:            deployment.kubernetes.io/revision: 1
<output_omitted>
```

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

```
apiVersion: extensions/v1beta1
kind: Deployment
Metadata:

<output_omitted>

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

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

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

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

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


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

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

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

....
        resources:
          limits:
            memory: 4Gi
          requests:
            memory: 2500Mi
        terminationMessagePath: /dev/termination-log
....
```

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


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

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

student@lfs458-node-1a0a:~$ kubectl logs hog-64cbfcc7cf-lwq66

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

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

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

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

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.

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

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

Only if top does not show high usage

Should the resources not show increased use, there may have been an issue inside of the container. Kubernetes may show it as running, but the actual workload has failed. Or the container may have failed; for example if you were missing a parameter the container may `panic`.

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

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


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

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

goroutine 1 [running]:
panic(0x5ff9a0, 0xc820014cb0)
        /usr/local/go/src/runtime/panic.go:481 +0x3e6
k8s.io/kubernetes/pkg/api/resource.MustParse(0x7ffe460c0e69, 0x5, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0)
        /usr/local/google/home/vishnuk/go/src/k8s.io/kubernetes/pkg/api/resource/quantity.go:134 +0x287
main.main()
        /usr/local/google/home/vishnuk/go/src/github.com/vishh/stress/main.go:24 +0x43
```

Here is an example of an improper parameter. The container is running, but not allocating memory. It should show the usage requested from the YAML file.

```
student@lfs458-node-1a0a:~$ kubectl get po
NAME                     READY     STATUS    RESTARTS   AGE
hog-1603763060-x3vnn     1/1       Running   0          8s


student@lfs458-node-1a0a:~$ kubectl logs hog-1603763060-x3vnn
I0927 21:09:23.514921       1 main.go:26] Allocating "0" memory, in "4ki" chunks, with a 1ms sleep \
                              between allocations
I0927 21:09:23.514984       1 main.go:39] Spawning a thread to consume CPU
I0927 21:09:23.514991       1 main.go:39] Spawning a thread to consume CPU
I0927 21:09:23.514997       1 main.go:29] Allocated "0" memory
```

Exercise 4.2: Resource Limits for a Namespace

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

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

```
student@lfs458-node-1a0a:~$ kubectl create namespace low-usage-limit
namespace/low-usage-limit created

student@lfs458-node-1a0a:~$ kubectl get namespace
NAME              STATUS    AGE
default           Active    1h
kube-node-lease   Active    1h
kube-public       Active    1h
kube-system       Active    1h
low-usage-limit   Active    42s
```

2. Create a YAML file which limits CPU and memory usage. The kind to use is `LimitRange`.

```
student@lfs458-node-1a0a:~$ vim low-resource-range.yaml
```

low-resource-range.yaml

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: low-resource-range
spec:
  limits:
  - default:
      cpu: 1
      memory: 500Mi
    defaultRequest:
```

```yaml
      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@lfs458-node-1a0a:~$ kubectl --namespace=low-usage-limit  \
     create -f low-resource-range.yaml
```

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

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

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

```
No resources found.
```

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

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

5. Create a new deployment in the namespace.

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

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

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

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



```
NAMESPACE          NAME            READY   UP-TO-DATE   AVAILABLE   AGE
default            hog             1/1     1            1           19m
kube-system        calico-typha    0/0     0            0           4h
kube-system        coredns         2/2     2            2           4h
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@lfs458-node-1a0a:~$ kubectl -n low-usage-limit get pods

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

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

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

<output_omitted>
spec:
  containers:
  - image: vish/stress
    imagePullPolicy: Always
    name: stress
    resources:
      limits:
        cpu: "1"
        memory: 500Mi
      requests:
        cpu: 500m
        memory: 100Mi
    terminationMessagePath: /dev/termination-log
<output_omitted>
```

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

```
student@lfs458-node-1a0a:~$ cp hog.yaml hog2.yaml

student@lfs458-node-1a0a:~$ vim hog2.yaml
```

hog2.yaml

```yaml
....
  labels:
    app: hog
  name: hog
  namespace: low-usage-limit     #<<--- Add this line, delete following
  selfLink: /apis/extensions/v1beta1/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@lfs458-node-1a0a:~$ kubectl create -f hog2.yaml

deployment.extensions/hog created
```

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

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

```
NAMESPACE          NAME           READY   UP-TO-DATE   AVAILABLE   AGE
default            hog            1/1     1            1           24m
kube-system        calico-typha   0/0     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@lfs458-node-1a0a:~$ kubectl -n low-usage-limit delete deployment hog

deployment.extensions "hog" deleted


student@lfs458-node-1a0a:~$ kubectl delete deployment hog

deployment.extensions "hog" deleted
```

Exercise 4.3: More Complex Deployment

> We will now deploy a more complex demo application to test the cluster. When completed it will be a sock shopping site. The short URL is shown below for:
> https://raw.githubusercontent.com/microservices-demo/microservices-demo/master/deploy/kubernetes/complete-demo.yaml

1. Begin by downloading the pre-made YAML file from github.


```
student@lfs458-node-1a0a:~$ wget https://tinyurl.com/y8bn2awp -O complete-demo.yaml
Resolving tinyurl.com (tinyurl.com)... 104.20.218.42, 104.20.219.42,
Connecting to tinyurl.com (tinyurl.com)|104.20.218.42|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://raw.githubusercontent.com/microservices-demo/microservices-...
--2017-11-02 16:54:27--  https://raw.githubusercontent.com/microservices-dem...
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.5...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101....
HTTP request sent, awaiting response... 200 OK
<output_omitted>
```

2. Find the expected namespaces inside the file. It should be `sock-shop`. Also note the various settings. This file will deploy several containers which work together, providing a shopping website. As we work with other parameters you could revisit this file to see potential settings.

```
student@lfs458-node-1a0a:~$ less complete-demo.yaml
```

complete-demo.yaml

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: carts-db
  labels:
    name: carts-db
  namespace: sock-shop
spec:
  replicas: 1
<output_omitted>
```

3. Create the namespace and verify it was made.

```
student@lfs458-node-1a0a:~$ kubectl create namespace sock-shop
```

```
namespace/sock-shop created
```

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

```
NAME              STATUS   AGE
default           Active   4h
kube-node-lease   Active   4h
kube-public       Active   4h
kube-system       Active   4h
low-usage-limit   Active   15m
sock-shop         Active   5s
```

4. View the images the new application will deploy.

```
student@lfs458-node-1a0a:~$ grep image complete-demo.yaml
```

```
        image: mongo
        image: weaveworksdemos/carts:0.4.8
        image: weaveworksdemos/catalogue-db:0.3.0
        image: weaveworksdemos/catalogue:0.3.5
        image: weaveworksdemos/front-end:0.3.12
        image: mongo
<output_omitted>
```

5. Create the new shopping website using the YAML file. Use the namespace you recently created. Note that the deployments match the images we saw in the file.

```
student@lfs458-node-1a0a:~$ kubectl apply -n sock-shop -f complete-demo.yaml
```

```
deployment "carts-db" created
service "carts-db" created
deployment "carts" created
service "carts" created
<output_omitted>
```

6. Using the proper namespace will be important. This can be set on a per-command basis or as a shell parameter. Note the first command shows no pods. We must remember to pass the proper namespace. Some containers may not have fully downloaded or deployed by the time you run the command.

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

No resources found.


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

NAME                            READY     STATUS              RESTARTS   AGE
carts-511261774-c4jwv           1/1       Running             0          71s
carts-db-549516398-tw9zs        1/1       Running             0          71s
catalogue-4293036822-sp5kt      1/1       Running             0          71s
catalogue-db-1846494424-qzhvk   1/1       Running             0          71s
front-end-2337481689-6s65c      1/1       Running             0          71s
orders-208161811-1gc6k          1/1       Running             0          71s
orders-db-2069777334-4sp01      1/1       Running             0          71s
payment-3050936124-2cn2l        1/1       Running             0          71s
queue-master-2067646375-vzq77   1/1       Running             0          71s
rabbitmq-241640118-vk3m9        0/1       ContainerCreating   0          71s
shipping-3132821717-lm7kn       0/1       ContainerCreating   0          71s
user-1574605338-24xrb           0/1       ContainerCreating   0          71s
user-db-2947298815-lx9kp        1/1       Running             0          71s
```

7. Verify the shopping cart is exposing a web page. Use the public IP address of your AWS node (not the one derived from the prompt) to view the page. Note the external IP is not yet configured. Find the `NodePort` service. First try port 80 then try port 30001 as shown under the `PORTS` column.

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

NAME           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
carts          ClusterIP   10.100.154.148   <none>        80/TCP           95s
carts-db       ClusterIP   10.111.120.73    <none>        27017/TCP        95s
catalogue      ClusterIP   10.100.8.203     <none>        80/TCP           95s
catalogue-db   ClusterIP   10.111.94.74     <none>        3306/TCP         95s
front-end      NodePort    10.98.2.137      <none>        80:30001/TCP     95s
```



```
orders          ClusterIP   10.110.7.215     <none>        80/TCP       95s
orders-db       ClusterIP   10.106.19.121    <none>        27017/TCP    95s
payment         ClusterIP   10.111.28.218    <none>        80/TCP       95s
queue-master    ClusterIP   10.102.181.253   <none>        80/TCP       95s
rabbitmq        ClusterIP   10.107.134.121   <none>        5672/TCP     95s
shipping        ClusterIP   10.99.99.127     <none>        80/TCP       95s
user            ClusterIP   10.105.126.10    <none>        80/TCP       95s
user-db         ClusterIP   10.99.123.228    <none>        27017/TCP    95s
```

8. Check to see which node is running the containers. Note that the webserver is answering on a node which is not hosting the all the containers. First we check the master, then the second node. The containers should have to do with **kube proxy** services and **calico**. The following is the **sudo docker ps** on both nodes. The output is truncated, you will see several lines of output per container.

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

CONTAINER ID        IMAGE
d6b7353e5dc5        weaveworksdemos/user@sha256:2ffccc332963c89e035fea52201012208bf62df4...
6c18f030f15b        weaveworksdemosshipping@sha256:983305c948fded487f4a4acdeab5f898e89d5...
baaa8d67ebef        weaveworksdemos/queue-master@sha256:6292d3095f4c7aeed8d863527f8ef6d7...
<output_omitted>
```

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

CONTAINER ID        IMAGE
9452559caa0d        weaveworksdemospayment@sha256:5ab1c9877480a018d4dda10d6dfa382776...
993017c7b476        weaveworksdemos/user-db@sha256:b43f0f8a76e0c908805fcec74d1ad7f4a...
1356b0548ee8        weaveworksdemos/orders@sha256:b622e40e83433baf6374f15e076b53893f...
<output_omitted>
```

9. View all the new deployments. There should be about 14.

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



```
NAMESPACE       NAME           DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
kube-system     calico-typha   0         0         0            0           4h
kube-system     coredns        2         2         2            2           4h
low-usage-limit limited-hog    1         1         1            1           33m
sock-shop       carts          1         1         1            1           6m44s
sock-shop       carts-db       1         1         1            1           6m44s
sock-shop       catalogue      1         1         1            1           6m44s
<output_omitted>
```

Basic Node Maintenance

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

1. Use the terminal on the second node to get a count of the current docker containers. It should be something like 30, plus a line for status counted by **wc**. The main system should have something like 26 running, plus a line of status.

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

26


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

30
```

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

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

NAME              STATUS   ROLES    AGE     VERSION
lfs458-worker     Ready    <none>   4h      v1.14.1
lfs458-node-1a0a  Ready    master   4h      v1.14.1
```

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

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

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

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

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

```
Taints:             <none>
Taints:             node.kubernetes.io/unschedulable:NoSchedule
```

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

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

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

There are pending nodes to be drained:
 lfs458-worker
error: pods with local storage (use --delete-local-data to override):
carts-55f7f5c679-ffkq2, carts-db-5c55874946-w728d, orders-7b69bf5686-vtkcn
```

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


```
student@lfs458-node-1a0a:~$ kubectl drain lfs458-worker \
     --ignore-daemonsets --delete-local-data
```

```
node/lfs458-worker already cordoned
WARNING: Ignoring DaemonSet-managed pods: calico-node-vndn7, kube-proxy-rjjls;
Deleting pods with local storage: carts-55f7f5c679-ppv7p,
carts-db-5c55874946-h42v2, orders-7b69bf5686-t82lz, orders-db-7bc46bdb98-x5zrl
pod/carts-db-5c55874946-h42v2 evicted
pod/orders-db-7bc46bdb98-x5zrl evicted
pod/catalogue-db-66ff5bbbf5-2wmx4 evicted
pod/catalogue-5764fdf6d-8gk96 evicted
pod/orders-7b69bf5686-t82lz evicted
pod/front-end-f99dbcb9c-92q4p evicted
pod/carts-55f7f5c679-ppv7p evicted
```

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

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

```
6
```

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

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

```
node/lfs458-worker uncordoned
```

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

```
Taints:             <none>
Taints:             <none>
```

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

```
6
```


8. As we clean up our sock shop let us see some differences between pods and deployments. Start with a list of the pods that are running in the `sock-shop` namespace.

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

```
NAME                          READY     STATUS    RESTARTS   AGE
carts-db-549516398-tw9zs      1/1       Running   0          6h
catalogue-4293036822-sp5kt    1/1       Running   0          6h
<output_omitted>
```

9. Delete a few resources using the pod name.

```
student@lfs458-node-1a0a:~$ kubectl -n sock-shop delete pod \
   catalogue-4293036822-sp5kt catalogue-db-1846494424-qzhvk \
   front-end-2337481689-6s65c orders-208161811-1gc6k \
   orders-db-2069777334-4sp01
```

```
pod "catalogue-4293036822-sp5kt" deleted
pod "catalogue-db-1846494424-qzhvk" deleted
<output_omitted>
```

10. Check the status of the pods. There should be some pods running for only a few seconds. These will have the same name-stub as the Pods you recently deleted. The `Deployment` controller noticed expected number of Pods was not proper, so created new Pods until the current state matches the Pod manifest.

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

```
NAME                             READY     STATUS        RESTARTS   AGE
catalogue-4293036822-mtz8m       1/1       Running       0          22s
catalogue-db-1846494424-16n2p    1/1       Running       0          22s
front-end-2337481689-6s65c       1/1       Terminating   0          6h
front-end-2337481689-80gwt       1/1       Running       0          22s
```

11. Delete some of the resources via deployments.

```
student@lfs458-node-1a0a:~$ kubectl -n sock-shop delete deployment \
       catalogue catalogue-db front-end orders
```



```
deployment.extensions "catalogue" deleted
deployment.extensions "catalogue-db" deleted
deployment.extensions "front-end" deleted
deployment.extensions "orders" deleted
```

12. Check and both the pods and deployments you removed have not been recreated.

```
student@lfs458-node-1a0a:~$ kubectl -n sock-shop get pods | grep catalogue

student@lfs458-node-1a0a:~$ kubectl -n sock-shop get deployment

NAME           DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
carts          1         1         1            1           71m
carts-db       1         1         1            1           71m
orders-db      1         1         1            1           71m
payment        1         1         1            1           71m
queue-master   1         1         1            1           71m
rabbitmq       1         1         1            1           71m
shipping       1         1         1            1           71m
user           1         1         1            1           71m
user-db        1         1         1            1           71m
```

13. Delete the rest of the deployments. When no resources are found, examine the output of the `docker ps` command. None of the `sock-shop` containers should be found. Use the same file we created with to delete all of the objects made. You will get some errors because we deleted a few deployments by hand.

```
student@lfs458-node-1a0a:~$ kubectl delete -f complete-demo.yaml

<output_omitted>
```


Chapter 5

APIs and Access

5.1 API Access

API Access

- API driven architecture
- API groups
- RESTful style
- Standard HTTP verbs
- Deprecation process not yet determined

RESTful

- Responds to typical `HTTP` verbs (`GET`, `POST`, `DELETE` ...)
- Allows easy interaction with other ecosystems
- Scripting and interaction with deployment tools
- User impersonation headers

Checking Access

- Several ways to authenticate
- `auth can-i` subcommand to query authorization
- Accepts `can-i` and `reconcile` arguments
- More in Security chapter to follow.

Optimistic Concurrency

- Currently leverage JSON
- `resourceVersion`
- Clients must handle `409 CONFLICT Errors`

5.2 Annotations

Using Annotations

- Distinct from `Labels`
- Non-identifying metadata
- Key/value maps
- Metadata otherwise held in exterior databases
- Useful for third-party automation

5.3 Working with A Simple Pod

Simple Pod

- Lowest compute unit of K8s
- Typically multiple containers grouped together
- Created from PodSpec
- Few required, many optional
  - `apiVersion`
    Must match existing API group
  - `kind`
    The type of object to create
  - `metadata`
    At least a name
  - `spec`
    What to create and parameters

5.4 kubectl and API

Manage API Resources with kubectl

- API exposed via RESTful interface
- Use **curl** to access and test
- Use verbose mode
- Leverages HTTP verbs

Access From Outside The Cluster

- Can use **curl** from outside cluster
- Must use SSL/TLS for secure access
- Information found in `~/.kube/config`
- View server information via **kubectl config view**

~/.kube/config

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdF.....
    server: https://10.128.0.3:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: LS0tLS1CRUdJTib.....
    client-key-data: LS0tLS1CRUdJTi....
```

Namespaces

- **Linux** kernel feature
  - Segregates system resources
  - Core functionality of containers
- API Object
  - Four namespaces to begin with:
    - `default`
    - `kube-node-lease`
    - `kube-public`
    - `kube-system`
  - **- -all-namespaces**

Working with Namespaces

```
$ kubectl get ns
$ kubectl create ns linuxcon
$ kubectl describe ns linuxcon
$ kubectl get ns/linuxcon -o yaml
$ kubectl delete ns/linuxcon
```

API Resources with kubectl

- All available via **kubectl**
- `kubectl [command] [type] [Name] [flag]`
- `kubectl help` for more information
- Abbreviated names

Additional Resource Methods

- Various Endpoints
- CLI `--help`
- Online documentation

5.5 Swagger and OpenAPI

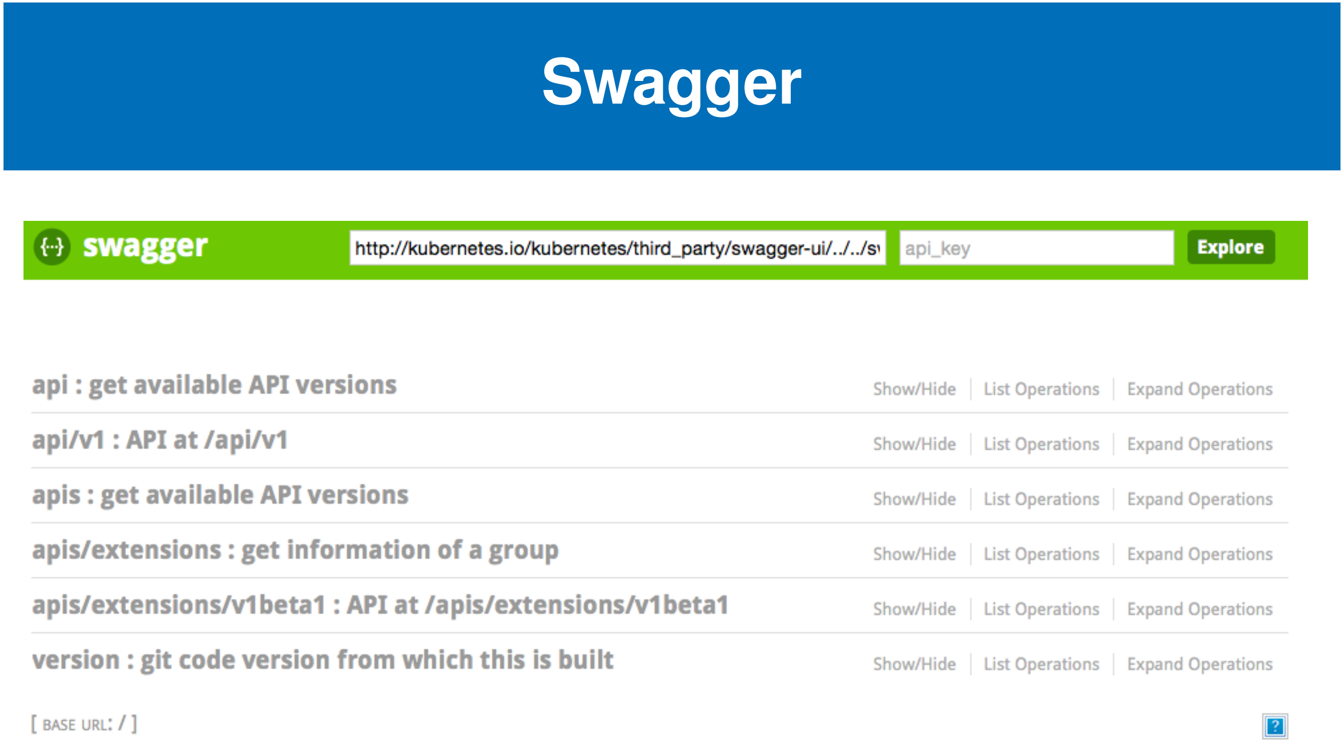

Figure 5.1: **Swagger Screenshot**

API Maturity

- Versioning of API levels for easier growth
- Not directly tied to software versioning
- Versions imply level of support
  - Alpha
  - Beta
  - Stable

5.6 Labs

Exercise 5.1: Configuring TLS Access

Overview

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

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

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

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

```
student@lfs458-node-1a0a:~$ less ~/.kube/config
<output_omitted>
```

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

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

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


```
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUM4akNDQWRxZ0F3SUJ
BZ0lJRy9wbC9rWEpNdmd3RFFZSktvWklodmNOQVFFTEJRQXdGVEVUTUJFR0
ExVUUKQXhNS2EzVmlaWEp1WlhSbGN6QWVGdzB4TnpFeU1UTXh0elEyTXpKY
```


UZ3MHhPREV5TVRNeE56UTJNelJhTURReApGekFWQmdOVkJBb1REbk41YzNS
```
<output_omitted>
```

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

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

student@lfs458-node-1a0a:~$ echo $key
<output_omitted>
```

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

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

student@lfs458-node-1a0a:~$ echo $auth
<output_omitted>
```

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

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

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

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

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

```
student@lfs458-node-1a0a:~$ kubectl config view |grep server
    server: https://10.128.0.3:6443
```

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

```
student@lfs458-node-1a0a:~$ curl --cert ./client.pem \
    --key ./client-key.pem \
    --cacert ./ca.pem \
    https://10.128.0.3:6443/api/v1/pods
```

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

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

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

```
{
    "kind": "Pod",
    "apiVersion": "v1",
    "metadata":{
        "name": "curlpod",
        "namespace": "default",
        "labels": {
            "name": "examplepod"
        }
    },
    "spec": {
        "containers": [{
            "name": "nginx",
            "image": "nginx",
            "ports": [{"containerPort": 80}]
        }]
```

```
    }
}
```

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@lfs458-node-1a0a:~$ curl --cert ./client.pem \
   --key ./client-key.pem --cacert ./ca.pem \
   https://10.128.0.3:6443/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@lfs458-node-1a0a:~$ 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.

```
student@lfs458-node-1a0a:~$ 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/10.128.0.3_6443. If you cannot find the lines, you may want to redirect all output to a file and grep for them. This information is cached, so you may see some differences should you run the command multiple times. As well your IP address may be different.

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

```
execve("/usr/bin/kubectl", ["kubectl", "get", "endpoints"], [/*....
....
openat(AT_FDCWD, "/home/student/.kube/cache/discovery/10.128.0.3_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@lfs458-node-1a0a:~$ cd /home/student/.kube/cache/discovery/
```

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

```
10.128.0.3_6443
```

```
student@lfs458-node-1a0a:~/.kube/cache/discovery$ cd 10.128.0.3_6443/
```

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

```
student@lfs458-node-1a0a:~/.kube/cache/discovery/10.128.0.3_6443$ ls
```

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


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

```
student@lfs458-node-1a0a:./10.128.0.3_6443$ find .
```

```
.
./events.k8s.io
./events.k8s.io/v1beta1
./events.k8s.io/v1beta1/serverresources.json
./apps
./apps/v1
./apps/v1/serverresources.json
./apps/v1beta1
./apps/v1beta1/serverresources.json
<output_omitted>
```

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

```
student@lfs458-node-1a0a:.$ python -m json.tool v1/serverresources.json
```

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


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

```
student@lfs458-node-1a0a:.$ python -m json.tool v1/serverresources.json | less
```

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

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

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

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



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

10. Looking at another file we find nine more.

```
student@lfs458-node-1a0a:$ python -m json.tool \
           apps/v1beta1/serverresources.json | grep kind
```

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

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

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

```
pod "curlpod" deleted
```

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

Chapter 6

API Objects

6.1 API Objects

Overview

- Ongoing growth, track release notes
- Discuss API objects:
  - `DaemonSets`, `ReplicaSets` now `apps/v1`
  - `StatefulSets` (once called `PetSets`) part of `v1` since v1.9
  - Jobs and CronJob now `batch/v1`
  - RBAC moved from v1alpha1 all the way to `v1` in one release
- Explain which API group contains these new API objects.
- Learn where to find additional resources to start using the new API objects.

6.2 The v1 Group

V1 API Group

- Pod
- Node
- Service Account
- Resource Quota
- Endpoint
- More added with each release

Discovering API Groups

```
$ curl https://localhost:6443/apis \
 --header "Authorization: Bearer $token" -k

{
  "kind": "APIGroupList",
  "apiVersion": "v1",
  "groups": [
    {
      "name": "apiregistration.k8s.io",
      "versions": [
        {
          "groupVersion": "apiregistration.k8s.io/v1beta1",
          "version": "v1beta1"
        }
      ],
      "preferredVersion": {
        "groupVersion": "apiregistration.k8s.io/v1beta1",
        "version": "v1beta1"
      }
```

6.3 API Resources

Deploying an Application

- Deployment
- ReplicaSet
- Pod

DaemonSets

- Ensures every node runs a single pod
- Similar to ReplicaSet
- Part of extension group

StatefulSet

- Similar to Deployment
- Ensures unique pods
- Guarantee ordering
- Stable in v1.9

Autoscaling

- Agents which add or remove resources from the cluster.
- Horizontal Pod Autoscaling (HPA)
  - Scale based on current CPU usage, or custom metric
  - Must have **Metrics Server** or custom component running
- Vertical Pod Autoscaler (under development)
- Cluster Autoscaler (CA)
  - Add or remove nodes based on utilization
  - Makes request to cloud provider
  - Pods which cannot be evicted prevent scale-down

Jobs

- Part of `Batch` API group
- Jobs run Pod until number of completions reached
  - Batch processing or one-off Pods
  - Ensure specified number of pods successfully terminate
  - Can run multiple Pods in parallel
- Cronjob to run Pod on regular basis
  - Creates a Pod about once per executing time
  - Some issues, job should be idempotent
  - Can run in serial or parallel
  - Same time syntax as Linux cron job

6.4 RBAC APIs

RBAC

- `rbac.authorization.k8s.io`
- Provide resources
  - `ClusterRole`
  - `ClusterRoleBinding`
  - `RoleBinding`
  - `Role`
- Combinded with quotas for typcial production deployments

6.5 Labs

Exercise 6.1: RESTful API Access

Overview

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

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

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

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

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: REDACTED
    server: https://10.128.0.3:6443
  name: kubernetes
<output_omitted>
```

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

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



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

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

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

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

```
student@lfs458-node-1a0a:~$ kubectl describe secret default-token-jdqp7
```

```
Name:          default-token-jdqp7
Namespace:     default
Labels:        <none>
<output_omitted>
token:      eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJrdWJlcm5ldGVz
L3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3Bh
Y2UiOiJkZWZhdWx0Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZWNyZXQubm
<output_omitted>
```

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

```
student@lfs458-node-1a0a:~$ export token=$(kubectl describe \
   secret default-token-jdqp7 |grep ^token |cut -f7 -d ' ')
```

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

```
student@lfs458-node-1a0a:~$ curl https://10.128.0.3:6443/apis \
         --header "Authorization: Bearer $token" -k
```



```
{
  "kind": "APIVersions",
  "versions": [
    "v1"
  ],
  "serverAddressByClientCIDRs": [
    {
      "clientCIDR": "0.0.0.0/0",
      "serverAddress": "10.128.0.3:6443"
    }
  ]
}
<output_omitted>
```


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

```
student@lfs458-node-1a0a:~$ curl https://10.128.0.3:6443/api/v1 \
         --header "Authorization: Bearer $token" -k
```

```
<output_omitted>
```

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

```
student@lfs458-node-1a0a:~$ curl \
         https://10.128.0.3:6443/api/v1/namespaces \
         --header "Authorization: Bearer $token" -k
```

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

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


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

Inside container

ls /var/run/secrets/kubernetes.io/serviceaccount/

ca.crt namespace token

exit

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@lfs458-node-1a0a:~$  kubectl proxy -h
Creates a proxy server or application-level gateway between localhost
and the Kubernetes API Server. It also allows serving static content
over specified HTTP path. All incoming data enters through one port
and gets forwarded to the remote kubernetes API Server port, except
for the path matching the static content path.

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

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

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.

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

<output_omitted>
```

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

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

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

Exercise 6.3: Working with Jobs

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

Create A Job

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

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

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


student@lfs458-node-1a0a:~$ kubectl get job
NAME     COMPLETIONS   DURATION   AGE
sleepy   0/1           3s         3s
```



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

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

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

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

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@lfs458-node-1a0a:~$ kubectl get jobs.batch sleepy -o yaml
<output_omitted>
  uid: c2c3a80d-d0fc-11e8-947a-42010a800002
spec:
  backoffLimit: 6
  completions: 1
  parallelism: 1
  selector:
    matchLabels:
<output_omitted>
```

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

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

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

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

student@lfs458-node-1a0a:~$ kubectl get jobs.batch
NAME     COMPLETIONS   DURATION   AGE
sleepy   0/5           5s         5s
```

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


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

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

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

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

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

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

job.batch "sleepy" deleted
```

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

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

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

```
student@lfs458-node-1a0a:~$ kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
sleepy-8xwpc                   1/1     Running   0          5s
sleepy-xjqnf                   1/1     Running   0          5s
<output_omitted>
```

```
student@lfs458-node-1a0a:~$ kubectl get jobs
NAME     COMPLETIONS   DURATION   AGE
sleepy   3/5           11s        11s
```

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

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

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

job.batch "sleepy" deleted


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

job.batch/sleepy created


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

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


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

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

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

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

<output_omitted>
status:
  conditions:
  - lastProbeTime: 2018-10-16T05:45:14Z
    lastTransitionTime: 2018-10-16T05:45:14Z
    message: Job was active longer than specified deadline
    reason: DeadlineExceeded
```

```
    status: "True"
    type: Failed
  failed: 2
  startTime: 2018-10-16T05:44:59Z
  succeeded: 3
```

14. Delete the job.

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

Create a CronJob

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

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

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

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

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

```yaml
apiVersion: batch/v1beta1    #<-- Add beta1 to be v1beta1
kind: CronJob                #<-- Update this line to CronJob
metadata:
  name: sleepy
spec:
```



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

student@lfs458-node-1a0a:~$ kubectl get cronjobs.batch
NAME     SCHEDULE      SUSPEND   ACTIVE   LAST SCHEDULE   AGE
sleepy   */2 * * * *   False     0        <none>          8s

student@lfs458-node-1a0a:~$ kubectl get jobs.batch
No resources found.
```

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

```
student@lfs458-node-1a0a:~$ kubectl get cronjobs.batch
NAME     SCHEDULE      SUSPEND   ACTIVE   LAST SCHEDULE   AGE
sleepy   */2 * * * *   False     0        21s             2m1s
```

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

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

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

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

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

```yaml
....
  jobTemplate:
    spec:
      template:
        spec:
          activeDeadlineSeconds: 10  #<-- Add this line
          containers:
          - name: resting
....
```

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@lfs458-node-1a0a:~$ kubectl delete cronjobs.batch sleepy

cronjob.batch "sleepy" deleted
```



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

cronjob.batch/sleepy created


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

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


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

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


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

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


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

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


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

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

7. Clean up by deleting the `CronJob`.

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

cronjob.batch "sleepy" deleted
```


Chapter 7

Managing State With Deployments

7.1 Deployment Overview

Overview

- Deployments
- Application Updates
- Labels
- `ReplicaSet`

7.2 Managing Deployment States

Deployments

- Client side rolling updates with **RC** (**Replication Controllers**)
- Deployments are server-side
- Generate `ReplicaSets`

```
$ kubectl create deployment dev-web --image=nginx:1.13.7-alpine

deployment "dev-web" created
```

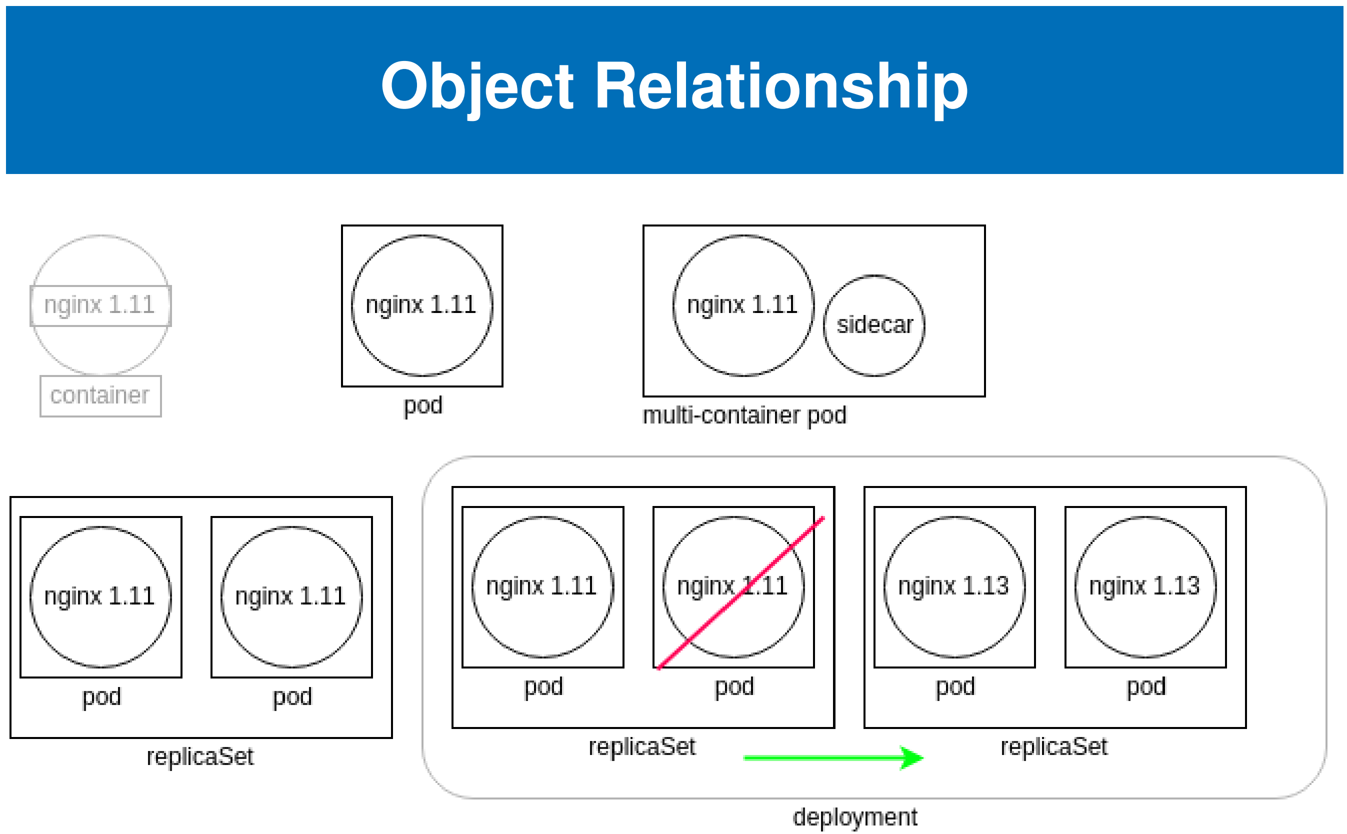

Figure 7.1: **Container to Deployment**

7.3 Deployments and Replica Sets

Deployment Details

- Generate YAML of newly created objects

  `kubectl get deployments,rs,pods -o yaml`

- Sometimes JSON output can make it more clear

  `kubectl get deployments,rs,pods -o json`

```
apiVersion: v1
items:
- apiVersion: extensions/v1beta1
  kind: Deployment
```

Deployment Configuration Metadata

```
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  creationTimestamp: 2017-12-21T13:57:07Z
  generation: 1
  labels:
    app: dev-web
  name: dev-web
  namespace: default
  resourceVersion: "774003"
  selfLink: /apis/extensions/v1beta1/namespaces/default/deployments/dev-web
  uid: d52d3a63-e656-11e7-9319-42010a800003
```

Deployment Configuration Spec

```
spec:
  progressDeadlineSeconds: 600
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: dev-web
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
```

Deployment Configuration Pod Template

```
template:
  metadata:
  creationTimestamp: null
    labels:
      app: dev-web
  spec:
    containers:
    - image: nginx:1.13.7-alpine
      imagePullPolicy: IfNotPresent
      name: dev-web
      resources: {}
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
    dnsPolicy: ClusterFirst
    restartPolicy: Always
    schedulerName: default-scheduler
    securityContext: {}
    terminationGracePeriodSeconds: 30
```


Deployment Configuration Status

```
status:
  availableReplicas: 2
  conditions:
  - lastTransitionTime: 2017-12-21T13:57:07Z
    lastUpdateTime: 2017-12-21T13:57:07Z
    message: Deployment has minimum availability.
    reason: MinimumReplicasAvailable
    status: "True"
    type: Available
  observedGeneration: 2
  readyReplicas: 2
  replicas: 2
  updatedReplicas: 2
```

Scaling and Rolling Updates

- Deployment configuration can be dynamically updated Controllers
- Use `set` argument to create new Replica Set
- Also use `edit` to trigger update
- Update YAML file and use `kubectl apply`

Deployment Rollbacks

- Previous ReplicaSets retained for rollback
- Can pause and resume
- Use `kubectl update` to change spec only

Deployment Rollbacks (cont.)

```
$ kubectl rollout pause deployment/ghost

$ kubectl rollout resume deployment/ghost
```

7.4 DaemonSets

Using DaemonSets

- Runs on every node
- Same image on each
- Added and removed dynamically
- Use `kind: DaemonSet`

7.5 Labels

Labels

- Can exist in every resource metadata
- Hash label created by default
- Immutable as of API version `apps/v1`

Labels (cont.)

- During creation or on the fly
- Easy to query or select

7.6 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` will will manage `ReplicaSets` for you. We will also work with another object called a `DaemonSet` which ensures a container is running on newly added node.

Then we will update the software in a container, view the revision history, and roll-back to a previous version.

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

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

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

No resources found.
```

2. Create a YAML file for a simple `ReplicaSet`. The `apiVersion` setting depends on the version of Kubernetes you are using. Versions 1.8 and beyond will use `apps/v1beta1`, then perhaps someday `apps/v1beta2` and then probably a stable `apps/v1`. We will use an older version of **nginx** then update to a newer version later in the exercise.

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


rs.yaml

```yaml
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: rs-one
spec:
  replicas: 2
  template:
    metadata:
      labels:
        system: ReplicaOne
    spec:
      containers:
      - name: nginx
        image: nginx:1.9.1
        ports:
        - containerPort: 80
```

3. Create the `ReplicaSet`:

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

4. View the newly created `ReplicaSet`:

```
student@lfs458-node-1a0a:~$ kubectl describe rs rs-one
Name:           rs-one
Namespace:      default
Selector:       system=ReplicaOne
Labels:         system=ReplicaOne
Annotations:    <none>
Replicas:       2 current / 2 desired
```

```
Pods Status:     2 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:        system=ReplicaOne
  Containers:
   nginx:
    Image:            nginx:1.9.1
    Port:             80/TCP
    Environment:      <none>
    Mounts:           <none>
  Volumes:            <none>
Events:               <none>
```

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

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

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

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

```
student@lfs458-node-1a0a:~$ kubectl delete rs rs-one --cascade=false

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

7. View the `ReplicaSet` and Pods again:

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

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


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

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

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

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

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

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

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

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

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

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

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

```
student@lfs458-node-1a0a:~$ kubectl edit po rs-one-3c6pb
```

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

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

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

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

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

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

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

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

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

replicaset.extensions "rs-one" deleted


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

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

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

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

No resources found.


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

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

15. Delete the remaining Pod using the label.


```
student@lfs458-node-1a0a:~$ kubectl delete po -l system=IsolatedPod
pod "rs-one-3c6pb" deleted
```

Exercise 7.2: Working with DaemonSets

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

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

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

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

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

ds.yaml

```yaml
....
kind: DaemonSet
....
  name: ds-one
....
```

```yaml
  replicas: 2 #<<<----Remove this line
....
        system: DaemonSetOne
....
```

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

```
student@lfs458-node-1a0a:~$ kubectl create -f ds.yaml
daemonset.extensions/ds-one created

student@lfs458-node-1a0a:~$ kubectl get ds
NAME     DESIRED   CURRENT   READY     UP-TO-DATE   AVAILABLE   NODE-SELECTOR   AGE
ds-one   2         2         2         2            2           <none>          1m

student@lfs458-node-1a0a:~$ kubectl get po
NAME                    READY     STATUS    RESTARTS   AGE
ds-one-b1dcv            1/1       Running   0          2m
ds-one-z31r4            1/1       Running   0          2m
```

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

```
student@lfs458-node-1a0a:~$ kubectl describe po ds-one-b1dcv | grep Image:
    Image:                      nginx:1.9.1
```

Exercise 7.3: Rolling Updates and Rollbacks

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

nginx versions

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

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

```
student@lfs458-node-1a0a:~$ kubectl get ds ds-one -o yaml \
      | grep -A 1 Strategy

  updateStrategy:
    type: OnDelete
```

2. Update the `DaemonSet` to use a newer version of the **nginx** server. This time use the **set** command instead of **edit**. Set the version to be `1.12.1-alpine`.

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

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

```
student@lfs458-node-1a0a:~$ kubectl describe po ds-one-b1dcv |grep Image:
    Image:                 nginx:1.9.1
```

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

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

student@lfs458-node-1a0a:~$ kubectl get po
NAME                   READY     STATUS    RESTARTS   AGE
ds-one-xc86w           1/1       Running   0          19s
ds-one-z31r4           1/1       Running   0          4m8s
```

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

    Image:                  nginx:1.12.1-alpine
```

5. View the image running on the older Pod. It should still show version 1.9.1.

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

    Image:                  nginx:1.9.1
```

6. View the history of changes for the `DaemonSet`. You should see two revisions listed. The number of revisions kept is set in the `DaemonSet` with v.1.12.1 the history kept has increased to ten from two, by default.

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

daemonsets "ds-one"
REVISION        CHANGE-CAUSE
1                <none>
2                <none>
```

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

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

daemonsets "ds-one" with revision #1
Pod Template:
  Labels:         system=DaemonSetOne
  Containers:
   nginx:
    Image:         nginx:1.9.1
    Port:         80/TCP
    Environment:        <none>
    Mounts:        <none>
  Volumes:        <none>

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



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

8. Use `kubectl rollout undo` to change the `DaemonSet` back to an earlier version. As we are still using the `OnDelete` strategy there should be no change to the Pods.

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

daemonset.extensions/ds-one rolled back


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

    Image:                nginx:1.12.1-alpine
```

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

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

pod "ds-one-xc86w" deleted


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

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


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

    Image:                nginx:1.9.1
```

10. View the details of the `DaemonSet`. The Image should be v1.9.1 in the output.

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



```
    Image:                  nginx:1.9.1
```

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

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

```
apiVersion: extensions/v1beta1
kind: DaemonSet
.....
      terminationGracePeriodSeconds: 30
  templateGeneration: 3
  updateStrategy:
    type: OnDelete
status:
  currentNumberScheduled: 2
.....
```

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

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

13. Edit the file. Change the name, around line eight and the update strategy around line 38.

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

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

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

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



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

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

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

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

    Image:               nginx:1.9.1
```

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

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

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

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

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

NAME     DESIRED   CURRENT   READY     UP-TO-DATE   AVAILABLE   NODE-SELECTOR   AGE
ds-two   2         2         2         2            2           <none>          10m
```

17. Now view the age of the Pods. Two should be much younger than the `DaemonSet`. They are also a few seconds apart due to the nature of the rolling update where one then the other pod was terminated and recreated.

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



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

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

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

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

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

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

20. View the changes in the update they should look the same as the previous history, but did not require the Pods to be deleted for the update to take place.

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

21. Clean up the system by removing one of the DaemonSets. We will leave the other running.

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


Chapter 8

Services

8.1 Overview

Overview

- Essential to micro-service architecture
- Connect Pods together, or outside cluster
- Service abstraction
- Load balancing
- Service types
- DNS resolution

Service Update Pattern

- Uses labels to match target Pods
- Rolling Deployments
- Traffic Shift

8.2 Accessing Services

Accessing an Application With A Service

```
$ kubectl expose deployment/nginx --port=80 --type=NodePort
$ kubectl get svc

NAME         CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   10.0.0.1     <none>        443/TCP   18h
nginx        10.0.0.112   <nodes>       80/TCP    5s

$ kubectl get svc nginx -o yaml

apiVersion: v1
kind: Service
...
spec:
    clusterIP: 10.0.0.112
    ports:
    - nodePort: 31230
...
```

Open browser http://Public-IP:31230

Service Types

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName

Services Diagram

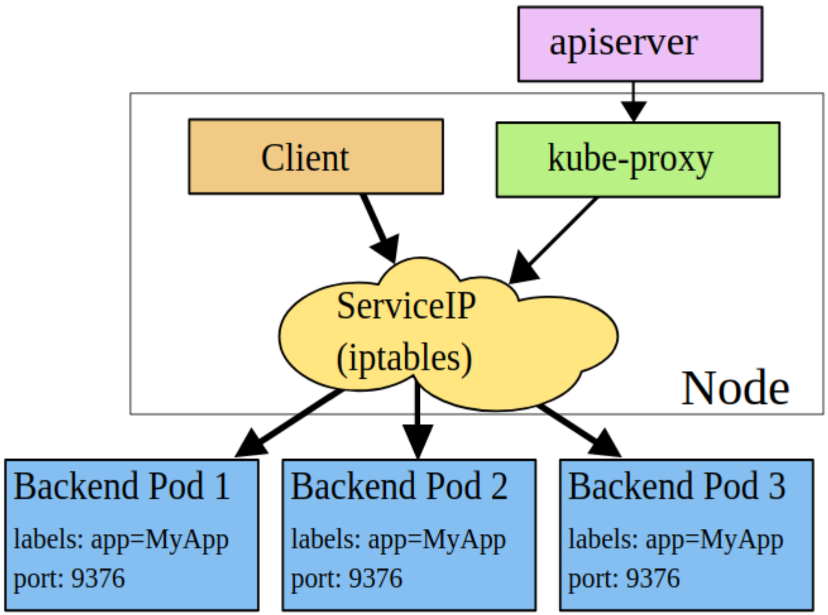

Figure 8.1: **Traffic from ClusterIP to Pod from** kubernetes.io

Local Proxy For Development

- Quick way to check service
- Available on localhost
- Not exposed to Internet

```
$ kubectl proxy

Starting to serve on 127.0.0.1:8001
```

8.3 DNS

DNS

- DNS provided as **CoreDNS** by default as of v1.13
- Server started for zones served
- Each loads plugin chains to provide services
  - tls
  - prometheus
  - health
  - errors

Verifying DNS Registration

```
$ kubectl exec -ti busybox -- nslookup nginx

Server: 10.0.0.10
Address 1: 10.0.0.10
Name: nginx
Address 1: 10.0.0.112
```

8.4 Labs

Exercise 8.1: Deploy A New Service

Overview

Services (also called **microservices**) are objects which declare a policy to access a logical set of Pods. They are typically assigned with `labels` to allow persistent access to a resource, when front or back end containers are terminated and replaced.

Native applications can use the `Endpoints` API for access. Non-native applications can use a Virtual IP-based bridge to access back end pods. `ServiceTypes Type` could be:

- **ClusterIP** default - exposes on a cluster-internal IP. Only reachable within cluster
- **NodePort** Exposes node IP at a static port. A ClusterIP is also automatically created.
- **LoadBalancer** Exposes service externally using cloud providers load balancer. `NodePort` and `ClusterIP` automatically created.
- **ExternalName** Maps service to contents of `externalName` using a `CNAME` record.

We use services as part of decoupling such that any agent or object can be replaced without interruption to access from client to back end application.

1. Deploy two **nginx** servers using **kubectl** and a new `.yaml` file. We will use the `v1beta` version of the API. The kind should be `Deployment` and label it with `nginx`. Create two replicas and expose port 8080. What follows is a well documented file. There is no need to include the comments when you create the file. This file can also be found among the other examples in the tarball.

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


nginx-one.yaml

```yaml
apiVersion: extensions/v1beta1
# 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:
  replicas: 2
# How many Pods of following containers to deploy
  template:
    metadata:
      labels:
        app: nginx
# 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:1.9.1
# The Docker image to deploy
        imagePullPolicy: Always
        name: nginx
# Unique name for each container, use local or Docker repo image
        ports:
```

```yaml
        - 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@lfs458-node-1a0a:~$ kubectl get nodes --show-labels
<output_omitted>
```

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

```
student@lfs458-node-1a0a:~$ kubectl create -f nginx-one.yaml
Error from server (NotFound): error when creating
"nginx-one.yaml": namespaces "accounting" not found
```

4. Create the namespace and try to create the deployment again. There should be no errors this time.

```
student@lfs458-node-1a0a:~$ kubectl create ns accounting
namespace/accounting" created

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

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

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

```
NAME                          READY    STATUS    RESTARTS   AGE
nginx-one-74dd9d578d-fcpmv    0/1      Pending   0          4m
nginx-one-74dd9d578d-r2d67    0/1      Pending   0          4m
```

6. View the node each has been assigned to (or not) and the reason, which shows under events at the end of the output.

```
student@lfs458-node-1a0a:~$ kubectl -n accounting describe pod \
      nginx-one-74dd9d578d-fcpmv
```

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

<output_omitted>

Events:
  Type     Reason            Age                 From       ....
  ----     ------            ----                ----
  Warning  FailedScheduling  37s (x25 over 2m29s)  default-scheduler
0/2 nodes are available: 2 node(s) didn't match node selector.
```

7. Label the secondary node. Verify the labels.

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

```
node/lfs458-worker labeled
```

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

```
NAME                STATUS   ROLES    AGE    VERSION   LABELS
lfs458-node-1a0a    Ready    master   1d1h   v1.12.1   \
     beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/
hostname=lfs458-node-1a0a,node-role.kubernetes.io/master=
lfs458-worker       Ready    <none>   1d1h   v1.12.1    \
     beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/
hostname=lfs458-worker,system=secondOne
```

8. View the pods in the `accounting` namespace. They may still show as `Pending`. Depending on how long it has been since you attempted deployment the system may not have checked for the label. If the Pods show `Pending` after a minute delete one of the pods. They should both show as `Running` after as a deletion. A change in state will cause the `Deployment` controller to check the status of both Pods.

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

NAME                          READY     STATUS    RESTARTS   AGE
nginx-one-74dd9d578d-fcpmv    1/1       Running   0          10m
nginx-one-74dd9d578d-sts5l    1/1       Running   0          3s
```

9. View Pods by the label we set in the YAML file. If you look back the Pods were given a label of `app=nginx`.

```
student@lfs458-node-1a0a:~$ kubectl get pods -l app=nginx --all-namespaces

NAMESPACE   NAME                          READY    STATUS    RESTARTS   AGE
accounting  nginx-one-74dd9d578d-fcpmv  1/1      Running   0          20m
accounting  nginx-one-74dd9d578d-sts5l  1/1      Running   0          9m
```

10. Recall that we exposed port 8080 in the YAML file. Expose the new deployment.

```
student@lfs458-node-1a0a:~$ kubectl -n accounting expose deployment nginx-one

service/nginx-one exposed
```

11. View the newly exposed endpoints. Note that port 8080 has been exposed on each Pod.

```
student@lfs458-node-9q6r:~$ kubectl -n accounting get ep nginx-one

NAME        ENDPOINTS                                  AGE
nginx-one   192.168.1.72:8080,192.168.1.73:8080        47s
```

12. Attempt to access the Pod on port 8080, then on port 80. Even though we exposed port 8080 of the container the application within has not been configured to listen on this port. The **nginx** server will listens on port 80 by default. A `curl` command to that port should return the typical welcome page.

```
student@lfs458-node-1a0a:~$ curl 192.168.1.72:8080
```

```
curl: (7) Failed to connect to 192.168.1.72 port 8080: Connection refused
```

```
student@lfs458-node-1a0a:~$ curl 192.168.1.72:80
```

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

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

```
student@lfs458-node-1a0a:~$ kubectl -n accounting delete deploy nginx-one
```

```
deployment.extensions "nginx-one" deleted
```

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

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

```
deployment.extensions/nginx-one created
```

Exercise 8.2: Configure a NodePort

In a previous exercise we deployed a `LoadBalancer` which deployed a `ClusterIP` and`NodePort` automatically. In this exercise we will deploy a `NodePort`. While you can access a container from within the cluster, one can use a `NodePort` to NAT traffic from outside the cluster. One reason to deploy a `NodePort` instead, is that a `LoadBalancer` is also a load balancer resource from cloud providers like GKE and AWS.

1. In a previous step we were able to view the **nginx** page using the internal Pod IP address. Now expose the deployment using the `--type=NodePort`. We will also give it an easy to remember name and place it in the `accounting` namespace. We could pass the port as well, which could help with opening ports in the firewall.

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



```
service/service-lab exposed
```

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

```
student@lfs458-node-1a0a:~$ kubectl -n accounting describe services
```

```
....
NodePort:                  <unset>  32103/TCP
....
```

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

```
student@lfs458-node-1a0a:~$ kubectl cluster-info
```

```
Kubernetes master is running at https://10.128.0.3:6443
KubeDNS is running at https://10.128.0.3:6443/api/v1/namespaces/
kube-system/services/kube-dns/proxy
To further debug and diagnose cluster problems, use
'kubectl cluster-info dump'.
```

4. Test access to the **nginx** web server using the combination of master URL and `NodePort`.

```
student@lfs458-node-1a0a:~$ curl http://10.128.0.3:32103
```

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

5. Using the browser on your local system, use the public IP address you use to SSH into your node and the port. You should still see the **nginx** default page.

Exercise 8.3: Use Labels to Manage Resources

1. Try to delete all Pods with the `app=nginx` label, in all namespaces. You should receive an error as this function must be narrowed to a particular namespace. Then delete using the appropriate namespace.

```
student@lfs458-node-1a0a:~$ kubectl delete pods -l app=nginx \
           --all-namespaces
Error: unknown flag: --all-namespaces
<output_omitted>

student@lfs458-node-1a0a:~$ kubectl -n accounting delete pods -l app=nginx
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@lfs458-node-1a0a:~$ kubectl -n accounting get pods
NAME                         READY     STATUS    RESTARTS   AGE
nginx-one-74dd9d578d-ddt5r   1/1       Running   0          1m
nginx-one-74dd9d578d-hfzml   1/1       Running   0          1m
```

3. We also gave a label to the deployment. View the deployment in the accounting namespace.

```
student@lfs458-node-1a0a:~$ kubectl -n accounting get deploy --show-labels
NAME      DESIRED CURRENT UP-TO-DATE AVAILABLE AGE LABELS
nginx-one 2       2       2          2         27m system=secondary
```

4. Delete the deployment using its label.

```
student@lfs458-node-1a0a:~$ kubectl -n accounting delete deploy \
                 -l system=secondary
deployment.extensions/nginx-one deleted
```

5. Remove the label from the secondary node. Note that the syntax is a minus sign directly after the key you want to remove, or `system` in this case.

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

Chapter 9

Volumes and Data

9.1 Volumes Overview

Overview

- Object to save data longer than container lifetime
- Many `Volume` types to choose from
- Define Persistent Volumes (PV)
- Define Persistent Volume Claims (PVC)
- Create `Secrets`
- Create `ConfigMaps`

9.2 Volumes

Introducing Volumes

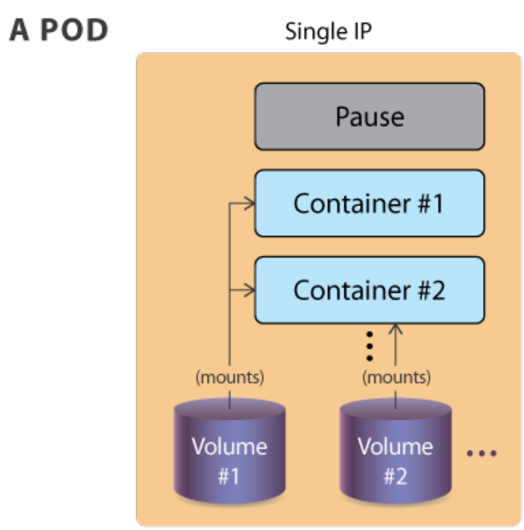

Figure 9.1: **K8s Pod Volumes**

Volume Spec

```
apiVersion: v1
kind: Pod
metadata:
    name: busybox
    namespace: default
spec:
    containers:
    - image: busybox
      name: busy
      command:
        - sleep
        - "3600"
      volumeMounts:
     - mountPath: /scratch
      name: scratch-volume
    volumes:
    - name: scratch-volume
            emptyDir: {}
```


Volume Types

- Several types possible, more being added
  - awsElasticBlockStore
  - azureDisk
  - azureFile
  - cephfs
  - csi
  - downwardAPI
  - emptyDir
  - fc (fibre channel)
  - flocker
  - gcePersistentDisk
  - gitRepo
  - glusterfs
  - hostPath
  - iscsi
  - local
  - nfs
  - projected
  - portworxVolume
  - quobyte
  - rbd
  - scaleIO
  - secret
  - storageos
  - vsphereVolume
  - persistentVolumeClaim

Shared Volume Example

```
 containers:
        - image: busybox
      volumeMounts:
        - mountPath: /busy
      name: test
      name: busy
        - image: busybox
      volumeMounts:
        - mountPath: /box
      name: test
      name: box
      volumes:
        - name: test
        emptyDir: {}

$ kubectl exec -ti busybox -c box -- touch /box/foobar
$ kubectl exec -ti busybox -c busy -- ls -l /busy total 0

-rw-r--r-- 1 root root 0 Nov 19 16:26 foobar
```


9.3 Persistent Volumes

Persistent Volumes and Claims

- Useful for porting data
- Resources managed via API
- Storage abstraction
- Several phases

```
$ kubectl get pv
$ kubectl get pvc
```

Persistent Volume

```yaml
kind: PersistentVolume
apiVersion: v1
metadata:
    name: 10Gpv01
    labels:
         type: local
spec:
    capacity:
        storage: 10Gi
    accessModes:
        - ReadWriteOnce
    hostPath:
        path: "/somepath/data01"
```

Persistent Volume Claim

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
    name: myclaim
spec:
    accessModes:
        - ReadWriteOnce
    resources:
        requests:
                storage: 8Gi
```

In the Pod:

```
spec:
    containers:
....
    volumes:
        - name: test-volume
          persistentVolumeClaim:
                claimName: myclaim
```

Dynamic Provisioning

- Claim filled via auto-provisioning
- No need for admin to pre-create PVs
- Uses the `StorageClass` API object
- `StorageClass` defines volume plugin, or provisioner to use
- Single, default class possible via annotation

9.4 Passing Data To Pods

Secrets

- Leverages `base64` encoding
- Is **not** encryption unless further configured
- Conversion to generic data, acceptable everywhere
- Encoded manually or via **kubectl create secret**

```
$ kubectl create secret generic mysql --from-literal=password=root
```

Using Secrets via Environment Variables

```
...
spec:
    containers:
    - image: mysql:5.5
      env:
      - name: MYSQL_ROOT_PASSWORD
        valueFrom:
            secretKeyRef:
                name: mysql
                key: password
          name: mysql
```

Mounting Secrets as Volumes

```
...
spec:
    containers:
    - image: busybox
      command:
        - sleep
        - "3600"
      volumeMounts:
      - mountPath: /mysqlpassword
        name: mysql
      name: busy
    volumes:
    - name: mysql
        secret:
                secretName: mysql
```

9.5 ConfigMaps

Portable Data With ConfigMaps

- Decouple configuration data from container image
- Not encoded or encrypted
- Can be created from various sources
  - Multiple files in same directory
  - Individual files
  - Literal values
- Can be consumed in various ways
  - Pod environmental variables from single or multiple ConfigMaps
  - Use ConfigMap values in Pod commands
  - Populate Volume from ConfigMap
  - Add ConfigMap data to specific path in Volume
  - Set file names and access mode in Volume from ConfigMap data
  - Can be used by system components and controllers

Using ConfigMaps

```
env:
- name: SPECIAL_LEVEL_KEY
  valueFrom:
    configMapKeyRef:
     name: special-config
     key: special.how

volumes:
    - name: config-volume
      configMap:
        name: special-config
```

9.6 Labs

Exercise 9.1: Create a ConfigMap

Overview

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

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

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

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

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

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

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

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

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

student@lfs458-node-1a0a:~$ echo y > primary/yellow

student@lfs458-node-1a0a:~$ echo k > primary/black

student@lfs458-node-1a0a:~$ echo "known as key" >> primary/black

student@lfs458-node-1a0a:~$ echo blue > favorite
```

2. Now we will create the `ConfigMap` and populate it with the files we created as well as a literal value from the command line.

```
student@lfs458-node-1a0a:~$ kubectl create configmap colors \
     --from-literal=text=black  \
     --from-file=./favorite  \
     --from-file=./primary/

configmap/colors created
```

3. View how the data is organized inside the cluster.

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

NAME      DATA      AGE
colors    6         30s
```

```
student@lfs458-node-1a0a:~$ kubectl get configmap colors -o yaml

apiVersion: v1
data:
  black: |
    k
    known as key
  cyan: |
    c
```

```
  favorite: |
    blue
  magenta: |
    m
  text: black
  yellow: |
    y
kind: ConfigMap
<output_omitted>
```

4. Now we can create a Pod to use the `ConfigMap`. In this case a particular parameter is being defined as an environment variable.

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

simpleshell.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: shell-demo
spec:
  containers:
  - name: nginx
    image: nginx
    env:
    - 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@lfs458-node-1a0a:~$ kubectl create -f simpleshell.yaml
pod/shell-demo created

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

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

6. All variables from a file can be included as environment variables as well. Comment out the previous `env:` stanza and add a slightly different `envFrom` to the file. Having new and old code at the same time can be helpful to see and understand the differences. Recreate the Pod, check all variables and delete the pod again. They can be found spread throughout the environment variable output.

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

simpleshell.yaml

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

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

pod/shell-demo created
```

```
student@lfs458-node-1a0a:~$ kubectl exec -it shell-demo \
           -- /bin/bash -c 'env'

HOSTNAME=shell-demo
NJS_VERSION=1.13.6.0.1.14-1~stretch
NGINX_VERSION=1.13.6-1~stretch
black=k
know as key

favorite=blue
<output_omitted>
```

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

pod "shell-demo" deleted
```

7. A `ConfigMap` can also be created from a YAML file. Create one with a few parameters to describe a car.

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

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

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

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

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

```
student@lfs458-node-1a0a:~$  kubectl exec -it shell-demo -- \
      /bin/bash -c 'df -ha |grep car'
```

```
/dev/sda1        20G  4.7G   15G  25% /etc/cars
```

```
student@lfs458-node-1a0a:~$ kubectl exec -it shell-demo -- \
    /bin/bash -c 'cat /etc/cars/car.trim'
```

```
Shelby  #Then your prompt
```

11. Delete the `Pod` and `ConfigMaps` we were using.

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

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



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

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

Exercise 9.2: Creating a Persistent NFS Volume (PV)

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

1. Install the software on your master node.

```
student@lfs458-node-1a0a:~$ sudo apt-get update && sudo \
             apt-get install -y nfs-kernel-server

<output_omitted>
```

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

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

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

student@lfs458-node-1a0a:~$ sudo bash -c \
               'echo software > /opt/sfw/hello.txt'
```

3. Edit the NFS server file to share out the newly created directory. In this case we will share the directory with all. You can always **snoop** to see the inbound request in a later step and update the file to be more narrow.

```
student@lfs458-node-1a0a:~$ sudo vim /etc/exports

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

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

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


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

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

student@lfs458-worker:~$ showmount -e lfs458-node-1a0a
Export list for lfs458-node-1a0a:
/opt/sfw  *

student@lfs458-worker:~$ sudo mount 10.128.0.3:/opt/sfw /mnt

student@lfs458-worker:~$ ls -l /mnt
total 4
-rw-r--r-- 1 root root 9 Sep 28 17:55 hello.txt
```

6. Return to the master node and create a YAML file for the object with `kind`, `PersistentVolume`. Use the hostname of the master server and the directory you created in the previous step. Only syntax is checked, an incorrect name or directory will not generate an error, but a Pod using the resource will not start. Note that the `accessModes` do not currently affect actual access and are typically used as labels instead.

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

PVol.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pvvol-1
spec:
  capacity:
    storage: 1Gi
```

```yaml
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  nfs:
    path: /opt/sfw
    server: lfs458-node-1a0a   #<-- Edit to match master node
    readOnly: false
```

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

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

student@lfs458-node-1a0a:~$ kubectl get pv
NAME      CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS
   CLAIM      STORAGECLASS   REASON    AGE
pvvol-1   1Gi        RWX           Retain          Available   4s
```

Exercise 9.3: Creating a Persistent Volume Claim (PVC)

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

1. Begin by determining if any currently exist.

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

2. Create a YAML file for the new pvc.

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

persistentvolumeclaim/pvc-one created


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

NAME      STATUS    VOLUME    CAPACITY   ACCESSMODES   STORAGECLASS   AGE
pvc-one   Bound     pvvol-1   1Gi        RWX                          4s
```

4. Look at the status of the pv again, to determine if it is in use. It should show a status of Bound.

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

NAME      CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS    CLAIM
   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 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.

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

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

nfs-pod.yaml

```yaml
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  generation: 1
  labels:
    run: nginx
  name: nginx-nfs
  namespace: default
  resourceVersion: "1411"
spec:
  replicas: 1
  selector:
    matchLabels:
      run: nginx
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
```



```yaml
  template:
    metadata:
      creationTimestamp: null
      labels:
        run: nginx
    spec:
      containers:
      - image: nginx
        imagePullPolicy: Always
        name: nginx
        volumeMounts:
        - name: nfs-vol
          mountPath: /opt
        ports:
        - containerPort: 80
          protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      volumes:                            #<<-- 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@lfs458-node-1a0a:~$ kubectl create -f nfs-pod.yaml
```

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

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

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

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

student@lfs458-node-1a0a:~$ kubectl describe pod nginx-nfs-1054709768-s8g28

Name:                  nginx-nfs-1054709768-s8g28
Namespace:          default
Node:                  lfs458-worker/10.128.0.5

<output_omitted>

    Mounts:
      /opt from nfs-vol (rw)

<output_omitted>

Volumes:
  nfs-vol:
    Type:        PersistentVolumeClaim (a reference to a PersistentV...
    ClaimName:        pvc-one
    ReadOnly:        false
<output_omitted>
```

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

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

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


Exercise 9.4: Using a ResourceQuota to Limit PVC Count and Usage

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

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

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

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

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

2. Create a yaml file for the `ResourceQuota` object. Set the storage limit to ten claims with a total usage of 500Mi.

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

storage-quota.yaml

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: storagequota
spec:
  hard:
    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@lfs458-node-1a0a:~$ kubectl create namespace small
namespace/small created
```

```
student@lfs458-node-1a0a:~$ kubectl describe ns small
Name:          small
Labels:        <none>
Annotations:   <none>
Status:        Active

No resource quota.

No resource limits.
```

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

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

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

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

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

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

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



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

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

No resource limits.
```

7. Remove the namespace line from the `nfs-pod.yaml` file. Should be around line 11 or so. This will allow us to pass other namespaces on the command line.

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

8. Create the container again.

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

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

9. Determine if the deployment has a running pod.

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

```
NAME        DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-nfs   1         1         1            0           43s
```

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

```
<output_omitted>
```


10. Look to see if the pods are ready.

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

NAME                         READY     STATUS    RESTARTS   AGE
nginx-nfs-2854978848-g3khf   1/1       Running   0          37s
```

11. Ensure the Pod is running and is using the NFS mounted volume. If you pass the namespace first `Tab` will auto-complete the pod name.

```
student@lfs458-node-1a0a:~$ kubectl -n small describe pod \
           nginx-nfs-2854978848-g3khf

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

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

12. View the quota usage of the namespace

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

<output_omitted>

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

No resource limits.
```

13. Create a 300M file inside of the `/opt/sfw` directory on the host and view the quota usage again. Note that with NFS the size of the share is not counted against the deployment.

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

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

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

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

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

```
301M        /opt/sfw
41M        /opt/cni/bin
41M        /opt/cni
341M        /opt/
```

14. Now let us illustrate what happens when a deployment requests more than the quota. Begin by shutting down the existing deployment.

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

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

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



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

15. Once the Pod has shut down view the resource usage of the namespace again. Note the storage did not get cleaned up when the pod was shut down.

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

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

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

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

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

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

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

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

```
NAME      CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS     CLAIM
STORAGECLASS    REASON    AGE
pvvol-1  1Gi    RWX    Retain    Released    small/pvc-one   44m
```

17. Dynamically provisioned storage uses the `ReclaimPolicy` of the `StorageClass` which could be `Delete`, `Retain`, or some types allow `Recycle`. Manually created persistent volumes default to `Retain` unless set otherwise at creation. The default storage policy is to retain the storage to allow recovery of any data. To change this begin by viewing the yaml output.

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



```yaml
....
    path: /opt/sfw
      server: lfs458-node-1a0a
  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@lfs458-node-1a0a:~$ kubectl delete pv/pvvol-1

persistentvolume "pvvol-1" deleted


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

  persistentVolumeReclaimPolicy: Retain


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

persistentvolume "pvvol-1" created
```

19. We will use `kubectl patch` to change the retention policy to `Delete`. The yaml output from before can be helpful in getting the correct syntax.

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

persistentvolume/pvvol-1 patched


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

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

20. View the current quota settings.

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

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

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

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

persistentvolumeclaim/pvc-one created


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

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

22. Remove the existing quota from the namespace.

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

NAME           CREATED AT
storagequota   2018-08-01T04:10:02Z


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

resourcequota "storagequota" deleted
```

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

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

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

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

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

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

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

No resource limits.
```

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

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

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

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

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

```
student@lfs458-node-1a0a:~$ kubectl -n small get po
NAME                        READY     STATUS     RESTARTS    AGE
nginx-nfs-2854978848-vb6bh  1/1       Running    0           58s
```


27. As we were able to deploy more pods even with apparent hard quota set, let us test to see if the reclaim of storage takes place. Remove the deployment and the persistent volume claim.

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

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

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

persistentvolumeclaim "pvc-one" deleted
```

28. View if the persistent volume exists. You will see it attempted a removal, but failed. If you look closer you will find the error has to do with the lack of a `deleter volume plugin` for NFS. Other storage protocols have a plugin.

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

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

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

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

persistentvolume "pvvol-1" deleted
```

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

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

YAML **PVol.yaml**

```
....
  persistentVolumeReclaimPolicy: Recycle
```

```
3 ....
```

31. Add a `LimitRange` to the namespace and attempt to create the persistent volume and persistent volume claim again. We can use the `LimitRange` we used earlier.

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

limitrange/low-resource-range created
```

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

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

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

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

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

persistentvolume/pvvol-1 created


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

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

34. Attempt to create the persistent volume claim again. The quota only takes effect if there is also a resource limit in effect.

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

```
Error from server (Forbidden): error when creating "pvc.yaml":
 persistentvolumeclaims "pvc-one" is forbidden: exceeded quota:
storagequota, requested: requests.storage=200Mi, used:
requests.storage=0, limited: requests.storage=100Mi
```

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

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

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

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

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

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

37. View the namespace settings.

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

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

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

student@lfs458-node-1a0a:~$ kubectl -n small get pvc
NAME      STATUS    VOLUME    CAPACITY   ACCESS MODES   STORAGECLASS   AGE
pvc-one   Bound     pvvol-1   1Gi        RWX                           7m

student@lfs458-node-1a0a:~$ kubectl -n small get pv
NAME    CAPACITY ACCESS MODES RECLAIM POLICY STATUS  CLAIM  STORA...
pvvol-1 1Gi      RWX    Recycle Bound small/pvc-one            ...
```

39. Delete the `pvc` and check the status of the `pv`. It should show as `Available`.

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

student@lfs458-node-1a0a:~$ kubectl -n small get pv
NAME    CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM STORA...
pvvol-1 1Gi      RWX          Recycle        Available       ...
```

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

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

Chapter 10

Ingress

10.1 Overview

Ingress Overview

- Single entrypoint to cluster
- Manage external access to services in cluster
  - HTTP
  - Load-balancing
  - SSL termination
  - Name-based virtual hosting
- GCE and nginx controllers currently supported
- HAProxy being developed

10.2 Ingress Controller

Ingress Controller

- Allow inbound traffic access to services
- Used instead of `NodePort`, or other services
- Proxy reconfigured according to rules

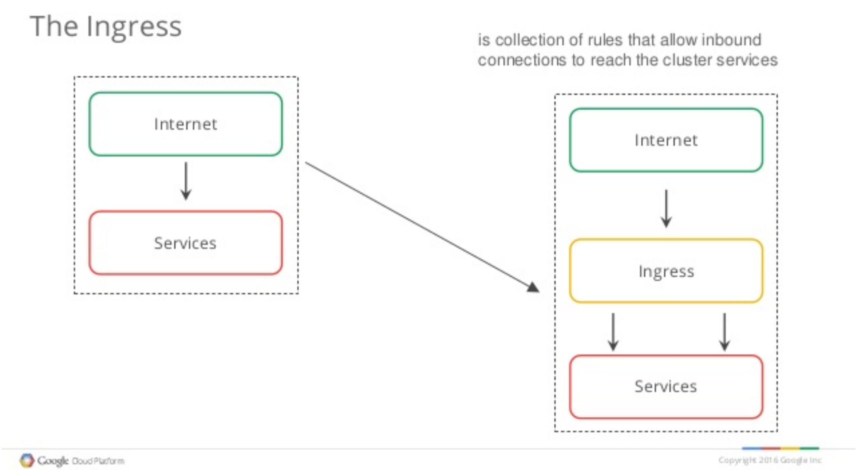

Figure 10.1: **Ingress Controller for inbound connections**

nginx

- Easy integration with RBAC
- Uses the annotation

  `kubernetes.io/ingress.class: "nginx"`
- L7 traffic requires the `proxy-real-ip-cidr` setting
- Bypasses kube-proxy to allow session affinity
- Does not use conntrack entries for iptables DNAT
- TLS requires `host` field to be defined

Google Load Balancer Controller (GLBC)

- Controller called **glbc**, must be created and started first
- Also create
  - Replication Controller with single replica
  - Three services for application Pod
  - Ingress with two hostnames and three endpoints for each service.
- Backend is group of virtual machine instances, Instance Group
- Multi-pool path:

```
Global Forwarding Rule -> Target HTTP Proxy -> URL map
-> Backend Service -> Instance Group
```

- Each pool checks next hop pool
- Not aware of GCE quota settings

Ingress API Resources

- An extension API
- POST to API server
- Manage like other resources

```
$ kubectl get ingress
$ kubectl delete ingress <ingress_name>
$ kubectl edit ingress <ingress_name>
```

Deploy Ingress Controller

- Several options available
- Firewall rules may stop traffic
- Default backend serves 404 pages

```
$ kubectl create -f backend.yaml
```

10.3 Ingress Rules

Creating an Ingress Rule

- Run deployment and expose port
- POST rules entry to controller

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
    name: ghost
spec:
    rules:
    - host: ghost.192.168.99.100.nip.io
      http:
      paths:
      - backend:
            serviceName: ghost
            servicePort: 2368
```

Multiple Rules

- Multiple Services can use multiple rules

```
rules:
- host: ghost.192.168.99.100.nip.io
  http:
    paths:
    - backend:
        serviceName: ghost
        servicePort: 2368
- host: nginx.192.168.99.100.nip.io
  http:
    paths:
    - backend:
        serviceName: nginx
        servicePort: 80
```

10.4 Labs

Exercise 10.1: Advanced Service Exposure

Configure an Ingress Controller

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

1. If you have a large number of services to expose outside of the cluster, or to expose a low-number port on the host node you can deploy an ingress controller or a service mesh. While **nginx** and **GCE** have controllers officially supported by Kubernetes.io, the **Traefik** ingress controller is easier to install. At the moment.

```
student@lfs458-node-1a0a:~$ kubectl create deployment secondapp \
   --image=nginx

deployment.apps/secondapp created
```

2. Find the labels currently in use by the deployment. We will use them to tie traffic from the ingress controller to the proper service.

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

  labels:
    app: secondapp
  name: secondapp
--
      labels:
        app: secondapp
    spec:
```

3. Expose the new server as a NodePort.


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

4. As we have RBAC configured we need to make sure the controller will run and be able to work with all necessary ports, endpoints and resources. Create a YAML file to declare a `clusterrole` and a `clusterrolebinding`.

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

ingress.rbac.yaml

```yaml
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: traefik-ingress-controller
rules:
  - apiGroups:
      - ""
    resources:
      - services
      - endpoints
      - secrets
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - extensions
    resources:
      - ingresses
    verbs:
      - get
      - list
```

```yaml
      - watch
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: traefik-ingress-controller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: traefik-ingress-controller
subjects:
- kind: ServiceAccount
  name: traefik-ingress-controller
  namespace: kube-system
```

5. Create the new role and binding.

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

clusterrole.rbac.authorization.k8s.io "traefik-ingress-controller" created
clusterrolebinding.rbac.authorization.k8s.io "traefik-ingress-controller" created
```

6. Create the Traefik controller. We will use a script directly from their website. This URL has a shorter version below: https://raw.githubusercontent.com/containous/traefik/v1.7/examples/k8s/traefik-ds.yaml

```
student@lfs458-node-1a0a:~$ wget https://bit.ly/2VCSz3s -O traefik-ds.yaml

<output_omitted>
2019-01-09 17:50:44 (188 MB/s) - ‘traefik-ds.yaml’ saved [1206/1206]
```

7. We need to take out some security context settings, such that the diff output between the new and old would be true. Add the `hostNetwork` line and remove the `securityContext` lines. The indentation for `hostNetwork` should line up with the `containers:` line.

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

#So that  diff traefik-ds.yaml.1 ds/traefik-ds.yaml reports this:

23a24              ##  Add this line
>       hostNetwork: true
34,39d34         ##  Remove these lines
<         securityContext:
<           capabilities:
<             drop:
<             - ALL
<             add:
<             - NET_BIND_SERVICE
```

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

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

9. Now that there is a new controller we need to pass some rules, so it knows how to handle requests. Note that the host mentioned is www.example.com, which is probably not your node name. We will pass a false header when testing. Also the service name needs to match the `secondapp` label we found in an earlier step.

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

YAML **ingress.rule.yaml**

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress-test
```

```yaml
  annotations:
    kubernetes.io/ingress.class: traefik
spec:
  rules:
  - host: www.example.com
    http:
      paths:
      - backend:
          serviceName: secondapp
          servicePort: 80
        path: /
```

10. Now ingest the rule into the cluster.

```
student@lfs458-node-1a0a:~$ kubectl create -f ingress.rule.yaml
ingress.extensions "ingress-test" created
```

11. We should be able to test the internal and external IP addresses, and see the nginx welcome page. The loadbalancer would present the traffic, a **curl** request in this case, to the externally facing interface. Use **ip a** to find the IP address of the interface which would face the loadbalancer. In this example the interface would be `ens4`, and the IP would be `10.128.0.7`.

```
student@lfs458-node-1a0a:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc mq state UP group default qlen 1000
    link/ether 42:01:0a:80:00:03 brd ff:ff:ff:ff:ff:ff
```



```
    inet 10.128.0.7/32 brd 10.128.0.3 scope global ens4
       valid_lft forever preferred_lft forever
<output_omitted>
```

```
student@lfs458-node-1a0a:~$ curl -H "Host: www.example.com" http://10.128.0.7/
```

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

```
student@lfs458-node-1a0a:~$ curl -H "Host: www.example.com" http://35.193.3.179
```

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

12. At this point we would keep adding more and more web servers. We'll configure one more, which would then be a process continued as many times as desired.

    Begin by deploying another nginx server. Give it a label and expose port 80.

    ```
    student@lfs458-node-1a0a:~$ kubectl create deployment thirdpage --image=nginx
    ```

    ```
    deployment.apps/thirdpage created
    ```

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

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

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

14. Expose the new server as a `NodePort`.

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

15. Now we will customize the installation. Run a bash shell inside the new pod. Your pod name will end differently. Install **vim** inside the container then edit the `index.html` file of nginx so that the title of the web page will be `Third Page`.

```
student@lfs458-node-1a0a:~$ kubectl exec -it thirdpage-5cf8d67664-zcmfh -- /bin/bash
```

Inside container

```
root@thirdpage-5cf8d67664-zcmfh:/\# apt-get update
```

<output_omitted >

```
root@thirdpage-5cf8d67664-zcmfh:/\#  apt-get install vim -y
```

<output_omitted >

```
root@thirdpage-5cf8d67664-zcmfh:/\# vim /usr/share/nginx/html/index.html
```

```
<!DOCTYPE html>
<html>
<head>
```

```
<title>Third Page</title>
<style>
```

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

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

```yaml
<output_omitted>
  - host: www.example.com
    http:
      paths:
      - backend:
          serviceName: secondapp
          servicePort: 80
        path: /
  - host: thirdpage.org
    http:
      paths:
      - backend:
          serviceName: thirdpage
          servicePort: 80
        path: /
 status:
<output_omitted>
```

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

```
student@lfs458-node-1a0a:~$ curl -H "Host: thirdpage.org" http://10.128.0.7/
```

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

Chapter 11

Scheduling

11.1 Overview

kube-scheduler

- Topology-aware algorithm to determine Pod placement
- Pod priority and preemption
- Labels used for each method
- **podAffinity** label encourage Pod on specific nodes
- Avoid via **podAntiAffinity**
- Node **taints** to repel specific Pods
- Pod **tolerations** of node **taints**
- **nodeSelector** to force specific Pods
- Require, prefer and evict

11.2 Scheduler Settings

Predicates

- **Predicates** check used to filter out inappropriate nodes
- Particular order from predicates.go

```
predicatesOrdering = []string{CheckNodeConditionPred,
GeneralPred, HostNamePred, PodFitsHostPortsPred,
MatchNodeSelectorPred, PodFitsResourcesPred, NoDiskConflictPred,
PodToleratesNodeTaintsPred, PodToleratesNodeNoExecuteTaintsPred,
CheckNodeLabelPresencePred, checkServiceAffinityPred,
MaxEBSVolumeCountPred, MaxGCEPDVolumeCountPred,
MaxAzureDiskVolumeCountPred, CheckVolumeBindingPred,
NoVolumeZoneConflictPred, CheckNodeMemoryPressurePred,
CheckNodeDiskPressurePred, MatchInterPodAffinityPred}
```

Priorities

- Set of **priority functions** used to weight resources
- Iteration of current status each adding or removing 1 to priority
- Highest value node chosen by scheduler
- `SelectorSpreadPriority` assigned to node with least pods
- View list at `master/pkg/scheduler/algorithm/priorities`
- **PriorityCLasses** can be used to allocate and preempt other pods
- Use a `Pod Disruption Budget` to limit disruption by preemption

11.3 Policies

Scheduling Policies

```
"kind" : "Policy",
"apiVersion" : "v1",
"predicates" : [
        {"name" : "MatchNodeSelector", "order": 6},
        {"name" : "PodFitsHostPorts", "order": 2},
        {"name" : "PodFitsResources", "order": 3},
        {"name" : "NoDiskConflict", "order": 4},
        {"name" : "PodToleratesNodeTaints", "order": 5},
        {"name" : "PodFitsHost", "order": 1}
        ],
"priorities" : [
        {"name" : "LeastRequestedPriority", "weight" : 1},
        {"name" : "BalancedResourceAllocation", "weight" : 1},
        {"name" : "ServiceSpreadingPriority", "weight" : 2},
        {"name" : "EqualPriority", "weight" : 1}
        ],
"hardPodAffinitySymmetricWeight" : 10
}
```

Pod Specification

- Pod Specification fields
  - `nodeName`
  - `nodeSelector`
  - `affinity`
  - `tolerations`
  - `schedulerName`

Specifying Node Label

- Match a label with `nodeSelector`
- Pod remains in `Pending` state until a suitable Node is found

```
spec:
  containers:
  - name: redis
    image: redis
  nodeSelector:
    net: fast
```

11.4 Affinity Rules

Pod Affinity Rules

- Uses `In`, `NotIn`, `Exists`, and `DoesNotExist` operators
- requiredDuringSchedulingIgnoredDuringExecution
- preferredDuringSchedulingIgnoredDuringExecution
- affinity
  - podAffinity
  - podAntiAffinity
- topologyKey

podAffinity Example

```
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: security
            operator: In
            values:
            - S1
        topologyKey: failure-domain.beta.kubernetes.io/zone
```

podAntiAffinity Example

```
podAntiAffinity:
  preferredDuringSchedulingIgnoredDuringExecution:
  - weight: 100
    podAffinityTerm:
      labelSelector:
        matchExpressions:
        - key: security
          operator: In
          values:
          - S2
      topologyKey: kubernetes.io/hostname
```

Node Affinity Rules

- Similar to Pod affinity rules, declared with affinity
- Uses `In`, `NotIn`, `Exists`, and `DoesNotExist` operators
- `requiredDuringSchedulingIgnoredDuringExecution`
- `preferredDuringSchedulingIgnoredDuringExecution`
- Planned for future
  - `requiredDuringSchedulingRequiredDuringExecution`

Node Affinity Example

```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: kubernetes.io/colo-tx-name
            operator: In
            values:
            - tx-aus
            - tx-dal
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: disk-speed
            operator: In
            values:
            - fast
            - quick
```

11.5 Taints and Tolerations

Taint

- Expressed as `key=value:`**effect**
- `key` and value `value` created by admin
- Effect must be **NoSchedule**, **PreferNoSchedule**, or **NoExecute**
- Only nodes can be tainted currently
- Multiple `taints` possible on a node

Tolerations

- Pod setting to run on `tainted` nodes
- Same **effects** as node taints
- Two operators
  - `Exists`
  - `Equal` which requires `value`

```
tolerations:
- key: "server"
  operator: "Equal"
  value: "ap-east"
  effect: "NoExecute"
  tolerationSeconds: 3600
```

Custom Scheduler

- Create and deploy custom scheduler as a container
- Run multiple schedulers simultaneously
- Pods can declare which scheduler to use
- Scheduler must be running or Pod remains `Pending`
- View scheduler and other information with **kubectl get events**

11.6 Labs

Exercise 11.1: Assign Pods Using Labels

Overview

While allowing the system to distribute Pods on your behalf is typically the best route, you may want to determine which nodes a Pod will use. For example you may have particular hardware requirements to meet for the workload. You may want to assign VIP Pods to new, faster hardware and everyone else to older hardware.

In this exercise we will use `labels` to schedule Pods to a particular node. Then we will explore `taints` to have more flexible deployment in a large environment.

1. Begin by getting a list of the nodes. They should be in the ready state and without added labels or taints.

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

NAME                STATUS    ROLES     AGE       VERSION
lfs458-node-1a0a    Ready     master    44h        v1.14.1
lfs458-worker       Ready     <none>    43h        v1.14.1
```

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

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

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


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

Taints:                         <none>
Taints:                         <none>
```


3. Verify there are no deployments running, outside of the `kube-system` namespace. If there are, delete them. Then get a count of how many containers are running on both the master and secondary nodes. There are about 24 containers running on the master in the following example, and eight running on the worker. There are status lines which increase the **wc** count. You may have more or less, depending on previous labs and cleaning up of resources.

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

NAMESPACE         NAME           DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
default           secondapp      1         1         1            1           37m
default           thirdpage      1         1         1            1           24m
kube-system       calico-typha   0         0         0            0           44h
kube-system       coredns        2         2         2            2           44h
low-usage-limit   limited-hog    1         1         1            1           24h
```

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

24
```

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

14
```

4. For the purpose of the exercise we will assign the master node to be VIP hardware and the secondary node to be for others.

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

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

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

node/lfs458-worker labeled
```

5. Verify your settings. You will also find there are some built in labels such as hostname, os and architecture type. The output below appears on multiple lines for readability.

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

```
NAME               STATUS   ROLES    AGE   VERSION   LABELS
lfs458-node-1a0a   Ready    master   44h   v1.14.1   beta.kubernetes.io/arch=
amd64,beta.kubernetes.io/os=linux,kubernetes.io/hostname=lfs458-node-1a0a,
node-role.kubernetes.io/master=,status=vip
lfs458-worker      Ready    <none>   44h   v1.14.1   beta.kubernetes.io/arch=
amd64,beta.kubernetes.io/os=linux,kubernetes.io/hostname=lfs458-worker,status=other
```

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

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

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
```

```yaml
      - "1000000"
    - name: vip4
      image: busybox
      args:
      - sleep
      - "1000000"
    nodeSelector:
      status: vip
```

7. Deploy the new pod. Verify the containers have been created on the master node. It may take a few seconds for all the containers to spawn. Check both the master and the secondary nodes.

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

student@lfs458-node-1a0a:~$ sudo docker ps |wc -l
29

student@lfs458-worker:~$ sudo docker ps |wc -l
8
```

8. Delete the pod then edit the file, commenting out the `nodeSelector` lines. It may take a while for the containers to fully terminate.

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

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



```
....
#  nodeSelector:
 #    status: vip
```

9. Create the pod again. Containers should now be spawning on either node. You may see pods for the `daemonsets` as well.

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

NAME                         READY   STATUS    RESTARTS   AGE
ds-one-bdqst                 1/1     Running   0          145m
ds-one-t2t7z                 1/1     Running   0          158m
secondapp-85765cd95c-2q9sx   1/1     Running   0          43m
thirdpage-7c9b56bfdd-2q5pr   1/1     Running   0          30m
```

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

pod/vip created
```

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

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

24
```

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

19
```

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

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

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

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


other.yaml

```yaml
....
  nodeSelector:
    status: other
```

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

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

student@lfs458-node-1a0a:~$ sudo docker ps |wc -l
24

student@lfs458-worker:~$ sudo docker ps |wc -l
24
```

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

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

student@lfs458-node-1a0a:~$ kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
ds-one-bdqst                   1/1     Running   0          153m
ds-one-t2t7z                   1/1     Running   0          166m
secondapp-85765cd95c-2q9sx     1/1     Running   0          51m
thirdpage-7c9b56bfdd-2q5pr     1/1     Running   0          40m
```

Exercise 11.2: Using Taints to Control Pod Deployment

Use taints to manage where Pods are deployed or allowed to run. In addition to assigning a Pod to a group of nodes, you may also want to limit usage on a node or fully evacuate Pods. Using taints is one way to achieve this. You may remember that the master node begins with a `NoSchedule` taint. We will work with three taints to limit or remove running pods.

1. Verify that the master and secondary node have the minimal number of containers running.

```
student@lfs458-node-1a0a:~$ kubectl delete deployment secondapp \
     thirdpage
```

```
deployment.extensions "secondapp" deleted
deployment.extensions "thirdpage" deleted
```

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

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

taint.yaml

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: taint-deployment
spec:
  replicas: 8
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name:  nginx
```



```yaml
        image: nginx:1.9.1
        ports:
        - containerPort: 80
```

3. Apply the file to create the deployment.

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

deployment.apps/taint-deployment created
```

4. Determine where the containers are running. In the following example three have been deployed on the master node and five on the secondary node. Remember there will be other housekeeping containers created as well. Your numbers may be slightly different.

```
student@lfs458-node-1a0a:~$ sudo docker ps |grep nginx

00c1be5df1e7        nginx@sha256:e3456c851a152494c3e.....
<output_omitted>


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

28


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

26
```

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

```
student@lfs458-node-1a0a:~$ kubectl delete deployment taint-deployment

deployment.extensions "taint-deployment" deleted


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

```
24
```

6. Now we will use a taint to affect the deployment of new containers. There are three taints, `NoSchedule`, `PreferNoSchedule` and `NoExecute`. The taints having to do with schedules will be used to determine newly deployed containers, but will not affect running containers. The use of `NoExecute` will cause running containers to move.

   Taint the secondary node, verify it has the taint then create the deployment again. We will use the key of `bubba` to illustrate the key name is just some string an admin can use to track Pods.

```
student@lfs458-node-1a0a:~$ kubectl taint nodes lfs458-worker \
          bubba=value:PreferNoSchedule

node/lfs458-worker tainted
```

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

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

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

deployment.apps/taint-deployment created
```

7. Locate where the containers are running. We can see that more containers are on the master, but there still were some created on the secondary. Delete the deployment when you have gathered the numbers.

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

32
```

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

22
```

```
student@lfs458-node-1a0a:~$ kubectl delete deployment taint-deployment

deployment.extensions "taint-deployment" deleted
```


8. Remove the taint, verify it has been removed. Note that the key is used with a minus sign appended to the end.

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

node/lfs458-worker untainted


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

Taints:                         <none>
Taints:                         <none>
```

9. This time use the `NoSchedule` taint, then create the deployment again. The secondary node should not have any new containers, with only daemonsets and other essential pods running.

```
student@lfs458-node-1a0a:~$ kubectl taint nodes lfs458-worker \
          bubba=value:NoSchedule

node/lfs458-worker tainted


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

deployment.apps/taint-deployment created


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

24


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

14
```

10. Remove the taint and delete the deployment. When you have determined that all the containers are terminated create the deployment again. Without any taint the containers should be spread across both nodes.

```
student@lfs458-node-1a0a:~$ kubectl delete deployment taint-deployment

deployment.extensions "taint-deployment" deleted
```



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

node/lfs458-worker untainted
```

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

deployment.apps/taint-deployment created
```

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

32
```

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

22
```

11. Now use the `NoExecute` to taint the secondary node. Wait a minute then determine if the containers have moved. The DNS containers can take a while to shutdown. A few containers will remain on the worker node to continue communication from the cluster.

```
student@lfs458-node-1a0a:~$ kubectl taint nodes lfs458-worker \
        bubba=value:NoExecute

node "lfs458-worker" tainted
```

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

32
```

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

6
```

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

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

node/lfs458-worker untainted
```



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

32
```

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

6
```

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

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

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

NAME                STATUS    ROLES     AGE       VERSION
lfs458-node-1a0a    Ready     master    44h        v1.14.1
lfs458-worker       Ready     <none>    44h        v1.14.1
```

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

node/lfs458-worker cordoned
error: DaemonSet-managed pods (use --ignore-daemonsets to ignore): kube-flannel-ds-fx3tx, kube-proxy-q2q4k
```

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

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

NAME               STATUS                      ROLES     AGE    VERSION
lfs458-node-1a0a   Ready                       master    44h    v1.14.1
lfs458-worker      Ready,SchedulingDisabled    <none>    44h    v1.14.1
```

15. Delete the deployment to destroy the current Pods.


```
student@lfs458-node-1a0a:~$ kubectl delete deployment taint-deployment
```

```
deployment.extensions "taint-deployment" deleted
```

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

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

```
deployment.apps/taint-deployment created
```

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

```
44
```

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

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

```
node/lfs458-worker uncordoned
```

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

```
NAME               STATUS    ROLES     AGE       VERSION
lfs458-node-1a0a   Ready     master    44h       v1.14.1
lfs458-worker      Ready     <none>    44h       v1.14.1
```

18. Delete and re-create the deployment.

```
student@lfs458-node-1a0a:~$ kubectl delete deployment taint-deployment
```

```
deployment.extensions "taint-deployment" deleted
```

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

```
deployment.apps/taint-deployment created
```


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

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

```
student@lfs458-node-1a0a:~$ kubectl delete deployment taint-deployment

deployment.extensions "taint-deployment" deleted
```

Chapter 12

Logging and Troubleshooting

12.1 Overview

Overview

- **Linux** troubleshooting via shell
- Turn on basic monitoring
- Set up cluster-wide logging
- External products **Fluentd**, **Prometheus** helpful.
- Internal **Metrics Server** and API

12.2 Troubleshooting Flow

Basic Steps

- Errors from command line
- Pod logs and state of the Pod
- Use shell to troubleshoot Pod DNS and network
- Check node logs for errors. Enough resources
- RBAC, SELinux or AppArmor security settings
- API calls to and from controllers to **kube-apiserver**
- Enable `auditing`
- Inter-node network issues, DNS and firewall
- Master server controllers.
  - Control Pods in pending or error state
  - Errors in log files
  - Enough resources

12.3 Basic Start Sequence

Cluster Start Sequence

- **systemd** starts `kubelet.service`
  - Uses `/etc/systemd/system/kubelet.service.d/10-kubeadm.conf`
  - Uses `/var/lib/kubelet/config.yaml` config file
  - **staticPodPath** set to `/etc/kubernetes/manifests/`
- **kubelet** creates all pods from *.yaml in directory
  - kube-apiserver
  - etcd
  - kube-controller-manager
  - kube-scheduler
- kube-controller-manager control loops use etcd data to start rest

12.4 Monitoring

Monitoring

- Enable the add-ons
- **Metrics Server** and API
- **Prometheus**

12.5 Logging

Logging Tools

- No cluster-wide logging in Kubernetes
- Often aggregated and digested by outside tools like **ElasticSearch**
- **Fluentd**
- **Kibana**

12.6 Troubleshooting Resources

More Resources

- Official documentation
- Major vendor pages
- Github pages
- Kubernetes **Slack** channel

12.7 Labs

Exercise 12.1: Review Log File Locations

Overview

In addition to various logs files and command output, you can use **journalctl** to view logs from the node perspective. We will view common locations of log files, then a command to view container logs. There are other logging options, such as the use of a **sidecar** container dedicated to loading the logs of another container in a pod.

Whole cluster logging is not yet available with Kubernetes. Outside software is typically used, such as **Fluentd**, part of `https://fluentd.org/`, which is another member project of `CNCF.io`, like Kubernetes.

Take a quick look at the following log files and web sites. As server processes move from node level to running in containers the logging also moves.

1. If using a **systemd**.based Kubernetes cluster, view the node level logs for **kubelet**, the local Kubernetes agent. Each node will have different contents as this is node specific.

```
student@lfs458-node-1a0a:~$ journalctl -u kubelet |less
```

```
<output_omitted>
```

2. Major Kubernetes processes now run in containers. You can view them from the container or the pod perspective. Use the **find** command to locate the **kube-apiserver** log. Your output will be different, but will be very long. Once you locate the files use the **diff** utility to compare them. There should be no difference, as they are symbolic links to `/var/log/pods/`. If you follow the links the log files are unique.

```
student@lfs458-node-1a0a:~$ sudo find / -name "*apiserver*log"
```

```
/var/log/containers/kube-apiserver-u16-12-1-dcb8_kube-system_kube-apiserver-
eddae7079382cd382cd55f8f46b192565dd16b6858206039d49b1ad4693c2a10.log
/var/log/containers/kube-apiserver-u16-12-1-dcb8_kube-system_kube-apiserver-
d00a48877af4ed4c7f8eedf2c7805c77cfabb31fcb453f7d89ffa52fc6ea5f36.log
```

```
student@lfs458-node-1a0a:~$ sudo diff /var/log/containers/kube-apiserver-u16-
12-1-dcb8_kube-system_kube-apiserver-eddae7079382cd382cd55f8f46b192565dd16b68
58206039d49b1ad4693c2a10.log    /var/log/containers/kube-apiserver-u16-12-1-
dcb8_kube-system_kube-apiserver-d00a48877af4ed4c7f8eedf2c7805c77cfabb31fcb453
f7d89ffa52fc6ea5f36.log
```

```
<output_omitted>
```

3. Take a look at the log file.

```
student@lfs458-node-1a0a:~$ sudo less /var/log/containers/kube-apiserver-u16-
12-1-dcb8_kube-system_kube-apiserver-d00a48877af4ed4c7f8eedf2c7805c77cfabb31f
cb453f7d89ffa52fc6ea5f36.log
```

```
<output_omitted>
```

4. Search for and review other log files for `kube-dns`, `kube-flannel`, and `kube-proxy`.

5. If **not** on a Kubernetes cluster using **systemd** which collects logs via **journalctl** you can view the text files on the master node.

   (a) `/var/log/kube-apiserver.log`
   Responsible for serving the API

   (b) `/var/log/kube-scheduler.log`
   Responsible for making scheduling decisions

   (c) `/var/log/kube-controller-manager.log`
   Controller that manages replication controllers

6. `/var/log/containers`

   Various container logs

7. `/var/log/pods/`

   More log files for current Pods.

8. Worker Nodes Files (on non-**systemd** systems)

   (a) `/var/log/kubelet.log`
   Responsible for running containers on the node

   (b) `/var/log/kube-proxy.log`
   Responsible for service load balancing

9. More reading: https://kubernetes.io/docs/tasks/debug-application-cluster/\debug-service/ and https://kubernetes.io/docs/tasks/debug-application-cluster/\determine-reason-pod-failure/

Exercise 12.2: Viewing Logs Output

> Container standard out can be seen via the **kubectl logs** command. If there is no standard out, you would not see any output. In addition, the logs would be destroyed if the container is destroyed.

1. View the current Pods in the cluster. Be sure to view Pods in all namespaces.

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

```
NAMESPACE     NAME                                        READY     STATUS    RESTARTS   AGE
default       ds-one-qc72k                                1/1       Running   0          3h
default       ds-one-z31r4                                1/1       Running   0          3h
....
kube-system   etcd-lfs458-node-1a0a                       1/1       Running   2          44h
kube-system   kube-apiserver-lfs458-node-1a0a             1/1       Running   2          44h
kube-system   kube-controller-manager-lfs458-node-1a0a    1/1       Running   2          44h
kube-system   kube-dns-2425271678-w80vx                   3/3       Running   6          44h
kube-system   kube-scheduler-lfs458-node-1a0a             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@lfs458-node-1a0a:~$ kubectl -n kube-system logs <Tab><Tab>
```

```
calico-etcd-n6h2q
etcd-lfs458-1-11-1update-cm35
calico-kube-controllers-74b888b647-9ds42
kube-apiserver-lfs458-1-11-1update-cm35
calico-node-6j8hc
kube-controller-manager-lfs458-1-11-1update-cm35
calico-node-dq6kf
kube-proxy-8sn6f
coredns-78fcdf6894-7fpfp
kube-proxy-wf5dr
coredns-78fcdf6894-g6k99
kube-scheduler-lfs458-1-11-1update-cm35
```

```
student@lfs458-node-1a0a:~$  kubectl -n kube-system logs \
          kube-apiserver-lfs458-1-11-1update-cm35
```

```
Flag --insecure-port has been deprecated, This flag will be
removed in a future version.
I0729 21:29:23.026394       1 server.go:703] external host
was not specified, using 10.128.0.2
I0729 21:29:23.026667       1 server.go:145] Version: v1.11.1
I0729 21:29:23.784000       1 plugins.go:158] Loaded 8 mutating
admission controller(s) successfully in the following order:
NamespaceLifecycle,LimitRanger,ServiceAccount,NodeRestriction,
Priority,DefaultTolerationSeconds,DefaultStorageClass,
MutatingAdmissionWebhook.
I0729 21:29:23.784025       1 plugins.go:161] Loaded 6 validating
admission controller(s) successfully in the following order:
LimitRanger,ServiceAccount,Priority,PersistentVolumeClaimResize,
ValidatingAdmissionWebhook,ResourceQuota.
<output_omitted>
```

3. View the logs of other Pods in your cluster.

Exercise 12.3: Adding tools for monitoring and metrics

With the deprecation of **Heapster** the new, integrated **Metrics Server** has been further developed and deployed. The **Prometheus** project of CNCF.io has matured from incubation to graduation, is commonly used for collecting metrics, and should be considered as well.

Configure Metrics

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

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

```
\textless output\_omitted \textgreater
```

2. Create the necessary objects.

```
student@lfs458-node-1a0a:~$ kubectl create -f metrics-server/deploy/1.8+/
```

```
clusterrole.rbac.authorization.k8s.io/system:aggregated-metrics-reader created
clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-delegator created
rolebinding.rbac.authorization.k8s.io/metrics-server-auth-reader created
apiservice.apiregistration.k8s.io/v1beta1.metrics.k8s.io created
serviceaccount/metrics-server created
deployment.extensions/metrics-server created
service/metrics-server created
clusterrole.rbac.authorization.k8s.io/system:metrics-server created
clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server created
```

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

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

```
\textless output\_omitted \textgreater
kube-proxy-ld2hb                         1/1     Running   0          2d21h
kube-scheduler-u16-1-13-1-2f8c           1/1     Running   0          2d21h
metrics-server-fc6d4999b-b9rjj           1/1     Running   0          42s
```

4. Edit the `metrics-server` deployment to allow insecure TLS the default certificate is x509 self-signed and not trusted by default. In production you may want to configure and replace the certificate.

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

```yaml
....
    spec:
      containers:
      - image: k8s.gcr.io/metrics-server-amd64:v0.3.1
        imagePullPolicy: Always
        name: metrics-server
        command:                                         #<-- Add these three lines
        - /metrics-server
        - --kubelet-insecure-tls
        resources: {}
....
```

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

```
student@lfs458-node-1a0a:~$ kubectl -n kube-system logs metrics-server<TAB>
```

```
I0110 20:21:27.333663       1 serving.go:273] Generated self-signed cert
(apiserver.local.config/certificates/apiserver.crt, apiserver.local.config
/certificates/apiserver.key)
[restful] 2019/01/10 20:21:28 log.go:33: [restful/swagger] listing is
available at https://:443/swaggerapi
[restful] 2019/01/10 20:21:28 log.go:33: [restful/swagger] https://:443/
```

```
swaggerui/ is mapped to folder /swagger-ui/
I0110 20:21:28.539435       1 serve.go:96] Serving securely on [::]:443
```

6. Test that the metrics working by viewing pod and node metrics. Your output may have different pods. It can take an minute or so for the metrics to populate and not return an error.

```
student@lfs458-node-1a0a:~$ kubectl top pods --all-namespaces

NAMESPACE     NAME                           CPU(cores)   MEMORY(bytes)
default       curlpod                        0m           2Mi
default       rs-one-7h2jq                   0m           2Mi
default       rs-one-n7qxc                   0m           2Mi
default       secondapp-ddd9845d6-qfbrs      0m           2Mi
kube-system   calico-node-594wc              27m          91Mi
kube-system   calico-node-sb2ft              21m          99Mi
\textless output\_omitted \textgreater
```

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

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

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

```
student@lfs458-node-1a0a:~$ curl --cert ./client.pem \
    --key ./client-key.pem --cacert ./ca.pem \
    https://10.142.0.3:6443/apis/metrics.k8s.io/v1beta1/nodes
```


```
{
  "kind": "NodeMetricsList",
  "apiVersion": "metrics.k8s.io/v1beta1",
  "metadata": {
    "selfLink": "/apis/metrics.k8s.io/v1beta1/nodes"
  },
```



```
  "items": [
    {
      "metadata": {
        "name": "u16-1-13-1-2f8c",
        "selfLink": "/apis/metrics.k8s.io/v1beta1/nodes/u16-1-13-1-2f8c",
        "creationTimestamp": "2019-01-10T20:27:00Z"
      },
      "timestamp": "2019-01-10T20:26:18Z",
      "window": "30s",
      "usage": {
        "cpu": "215675721n",
        "memory": "2414744Ki"
      }
    },
<output_omitted>
```


Configure the Dashboard

While the dashboard looks nice it has not been a common tool in use. Those that could best develop the tool tend to only use the CLI, so it may lack wanted functionality.

Compatability With Metric Server

The dashboard has not been updated to work with the **Metrics Server** now that **Heapster** has been deprecated. While there is some interest in getting the metrics to show in the dashboard there has been difficulty finding developers to work on the issue. https://github.com/kubernetes/dashboard/issues/2986

1. Create the dashboard. The short URL in the step below, which has an ”ell”, not number one, is for this longer URL: https://raw.githubusercontent.com/kubernetes/dashboard/master/aio/deploy/recommended/kubernetes-dashboard.yaml.

```
student@lfs458-node-1a0a:~$ kubectl create -f https://bit.ly/2G4e9Hu
```



```
secret/kubernetes-dashboard-certs created
serviceaccount/kubernetes-dashboard created
role.rbac.authorization.k8s.io/kubernetes-dashboard-minimal created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard-minimal created
deployment.apps/kubernetes-dashboard created
service/kubernetes-dashboard created
```

2. View the current services in all namespaces. Note that the `kubernetes-dashboard` is a `ClusterIP` and part of the `kube-system` namespace.

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

NAMESPACE     NAME                   TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)         AGE
default       kubernetes             ClusterIP   10.96.0.1        <none>        443/TCP         2d22h
default       secondapp              NodePort    10.97.83.161     <none>        80:32069/TCP    26h
default       thirdpage              ClusterIP   10.102.185.77    <none>        80/TCP          40h
kube-system   calico-typha           ClusterIP   10.101.192.117   <none>        5473/TCP        2d22h
kube-system   kube-dns               ClusterIP   10.96.0.10       <none>        53/UDP,53/TCP   2d22h
kube-system   kubernetes-dashboard   ClusterIP   10.107.224.246   <none>        443/TCP         29s
kube-system   metrics-server         ClusterIP   10.105.86.51     <none>        443/TCP         13m
```

3. Edit the `kubernetes-dashboard` and change the type to a `NodePort`.

```
student@lfs458-node-1a0a:~$ kubectl -n kube-system edit svc kubernetes-dashboard
```

```yaml
....
  selector:
    k8s-app: kubernetes-dashboard
  sessionAffinity: None
  type: NodePort                    #<-- Edit this line
status:
  loadBalancer: {}
```


4. Check the `kubernetes-dashboard` service again. The Type should show as `NodePort`. Take note of the high-numbered port, which is `30968` in the example below. Yours will be different.

```
student@lfs458-node-1a0a:~$ kubectl -n kube-system get svc kubernetes-dashboard

NAME                   TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)         AGE
kubernetes-dashboard   NodePort   10.107.224.246   <none>        443:30968/TCP   6m39s
```

5. There has been some issues with RBAC and the dashboard permissions to see objects. In order to ensure access to view various resources give the dashboard admin access.

```
student@lfs458-node-1a0a:~$ kubectl create clusterrolebinding kubernetes-dashboard \
    --clusterrole=cluster-admin --serviceaccount=kube-system:kubernetes-dashboard
```

6. On your local node open a browser and navigate to an HTTPS URL made of the `Public IP` and the high-numbered port. You will get a message about an insecure connection. Select the **Advanced** button, then **Add Exception...**, then **Confirm Security Exception**. The page should then show the `Kubernetes Dashboard`.

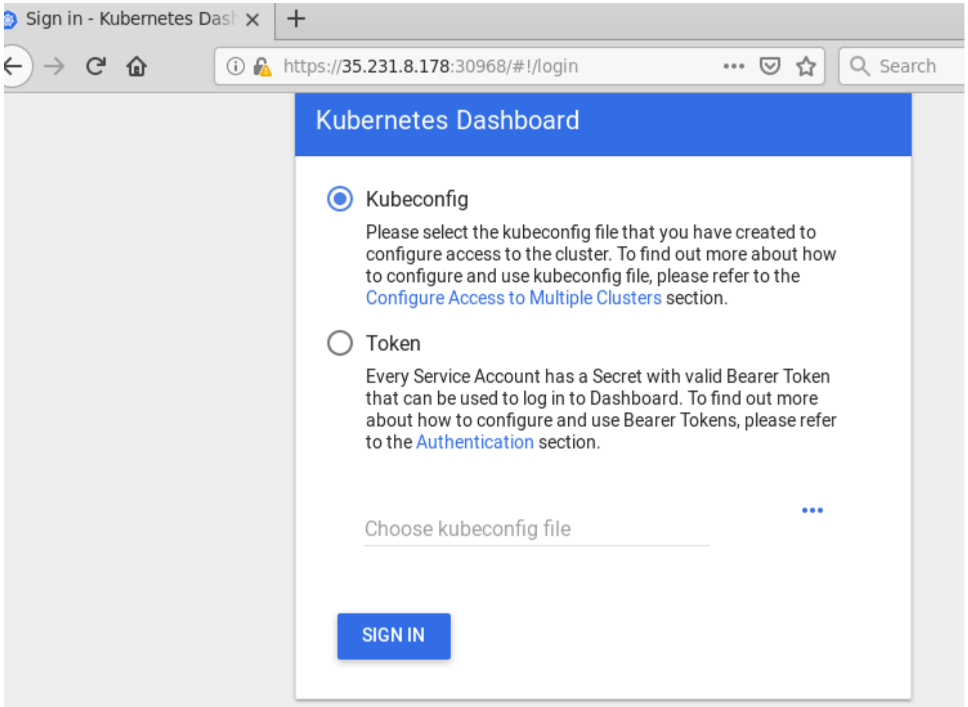

Figure 12.1: **External Access via Browser**

7. We will use the `Token` method to access the dashboard. With RBAC we need to use the proper token, the `kubernetes-dashboard-token` in this case. Find the token, copy it then paste into the login page. The **Tab** key can be helpful to complete the secret name instead of finding the hash.

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

```
....
Data
```



```
====
ca.crt:     1025 bytes
namespace:  11 bytes
token:      eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZX
JuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY
291bnQvc2VjcmV0Lm5hbWUiOiJrdWJlcm5ldGVzLWRhc2hib2FyZC10b2tlbi1wbW04NCIsImt1YmVybmV0ZXMuaW8vc2Vydmlj
ZWFjY291bnQvc2VydmljZS1hY2NvdW50Lm5hbWUiOiJrdWJlcm5ldGVzLWRhc2hib2FyZCIsImt1YmVybmV0ZXMuaW8vc2Vydml
jZWFjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6IjE5MDY4ZDIzLTE1MTctMTFlOS1hZmMyLTQyMDEwYThlMDAwMyIsInN1Yi
I6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDprdWJlLXN5c3RlbTprdWJlcm5ldGVzLWRhc2hib2FyZCJ9.aYTUMWr290pjt5i32rb8
qXpq4onn3hLhvz6yLSYexgRd6NYsygVUyqnkRsFE1trg9i1ftNXKJdzkY5kQzN3AcpUTvyj_BvJgzNh3JM9p7QMjI8LHTz4TrRZ
rvwJVWitrEn4VnTQuFVcADFD_rKB9FyI_gvT_QiW5fQm24ygTIgfOYd44263oakG8sL64q7UfQNW2wt5SOorMUtybOmX4CXNUYM8
G44ejEtv9GW5OsVjEmLIGaoEMX7fctwUN_XCyPdzcCg2WOxRHahBJmbCuLz2SSWL52q4nXQmhTq_L8VDDpt6LjEqXW6LtDJZGjVC
s2MnBLerQz-ZAgsVaubbQ
```


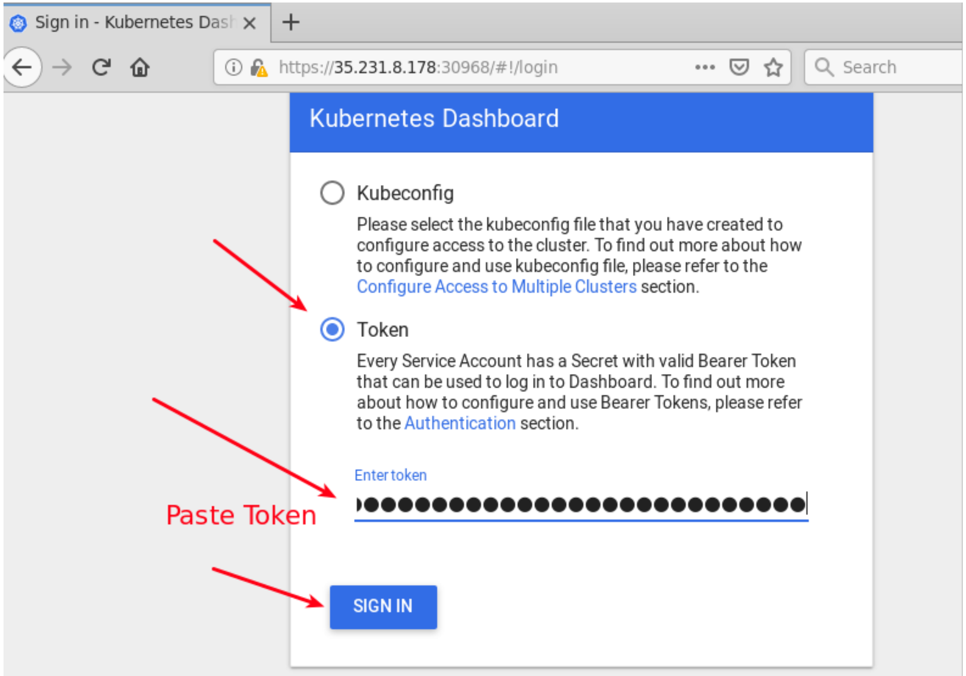

Figure 12.2: **External Access via Browser**

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

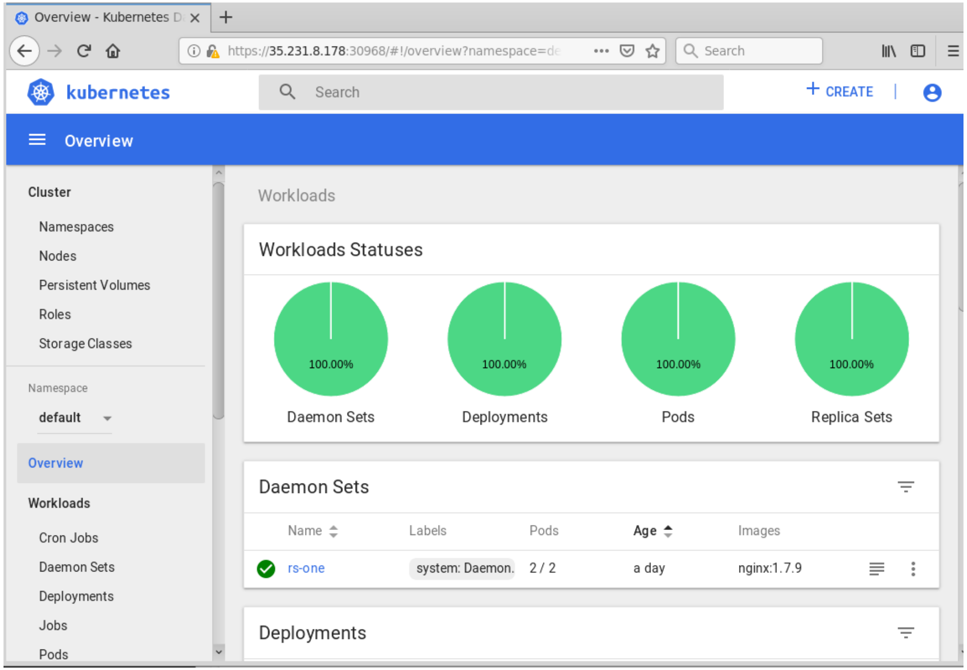

Figure 12.3: **External Access via Browser**

Chapter 13

Custom Resource Definition

13.1 Overview

Custom Resources

- Flexible to meet changing needs
- Create your own API objects
- Manage your API objects via **kubectl**
- Custom Resource Definitions (CRD)
- Aggregated APIs (AA)

13.2 Custom Resource Definitions

Custom Resource Definitions

- Easy to deploy
- Typically does not require programming
- Does not require another API server
- Namespaced or cluster-scoped

Configuration Example

```yaml
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: backups.stable.linux.com
spec:
  group: stable.linux.com
  version: v1
  scope: Namespaced
  names:
    plural: backups
    singular: backup
    shortNames:
    - bks
    kind: BackUp
```

New Object Configuration

```
apiVersion: "stable.linux.com/v1"
kind: BackUp
metadata:
  name: a-backup-object
spec:
  timeSpec: "* * * * */5"
  image: linux-backup-image
  replicas: 5
```

Optional Hooks

- Finalizer

```
metadata:
  finalizers:
  - finalizer.stable.linux.com
```

- Validation

```
validation:
    openAPIV3Schema:
      properties:
        spec:
          properties:
            timeSpec:
              type: string
              pattern: '^(\d+|\*)(/\d+)?(\s+(\d+|\*)(/\d+)?){4}$'
            replicas:
              type: integer
              minimum: 1
              maximum: 10
```

13.3 Aggregated APIs

Understanding Aggregated APIs (AA)

- Usually requires non-trivial programming
- More control over API behavior
- Subordinate API server behind primary
- Primary acts as proxy
- Leverages extension resource
- Mutual TLS auth between API servers
- RBAC rule to allow addition of API service objects

13.4 Labs

Exercise 13.1: Create a Custom Resource Definition

Overview

ThirdPartyResource is no longer included with the API in v1.8 and its use will return a validation error. If you have upgraded from a version prior to Kubernetes v1.7, you will need to convert them to CustomResourceDefinitions (CRD). A new resource often requires a controller to manage the resource. Creation of the controller is beyond the scope of this course, basically it is a watch-loop comparing a spec file to the current state and making changes until the states match. A good discussion of creating a controller can be found here: https://coreos.com/blog/introducing-operators.html.

We will make a simple CRD, but without any particular action. It will be enough to find the object ingested into the API and responding to commands.

1. We will create a new YAML file.

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

crd.yaml

```yaml
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: crontabs.training.lfs458.com
         # This name must match names below.
         # <plural>.<group> syntax
spec:
  scope: Cluster    #Could also be Namespaced
```

```yaml
  group: training.lfs458.com
  version: v1
  names:
    kind: CronTab      #Typically CamelCased for resource manifest
    plural: crontabs   #Shown in URL
    singular: crontab  #Short name for CLI alias
    shortNames:
    - ct               #CLI short name
```

2. Add the new resource to the cluster.

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

customresourcedefinition.apiextensions.k8s.io/crontabs.training.lfs458.com
created
```

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

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

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

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

Name:         crontabs.training.lfs458.com
Namespace:
Labels:       <none>
Annotations:  <none>
API Version:  apiextensions.k8s.io/v1beta1
Kind:         CustomResourceDefinition
<output_omitted>
```

4. Now that we have a new API resource we can create a new object of that type. In this case it will be a crontab-like image, which does not actually exist, but is being used for demonstration.

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

new-crontab.yaml

```yaml
apiVersion: "training.lfs458.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
```

5. Create the new object and view the resource using short and long name.

```
student@lfs458-node-1a0a:~$ kubectl create -f new-crontab.yaml
crontab.training.lfs458.com/new-cron-object created

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

NAME              AGE
new-cron-object   22s

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

NAME              AGE
new-cron-object   29s
```

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

```
Name:         new-cron-object
Namespace:
Labels:       <none>

<output_omitted>

Spec:
  Cron Spec:  */5 * * * *
  Image:      some-cron-image
Events:       <none>
```

6. To clean up the resources we will delete the CRD. This should delete all of the endpoints and objects using it as well.

```
student@lfs458-node-1a0a:~$ kubectl delete -f crd.yaml
```

```
customresourcedefinition.apiextensions.k8s.io
"crontabs.training.lfs458.com" deleted
```

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

```
Error from server (NotFound): Unable to list "crontabs": the server
could not find the requested resource
(get crontabs.training.lfs458.com)
```


Chapter 14

Kubernetes Federation

14.1 Overview

Cluster Federation

- Multiple, independent clusters
- Federated resources
- virtual-IP and DNS record auto discovery
- Higher availability
- Data locality
- Vendor-lock avoidance
- Bandwidth costs and complexity

14.2 Federated Resources

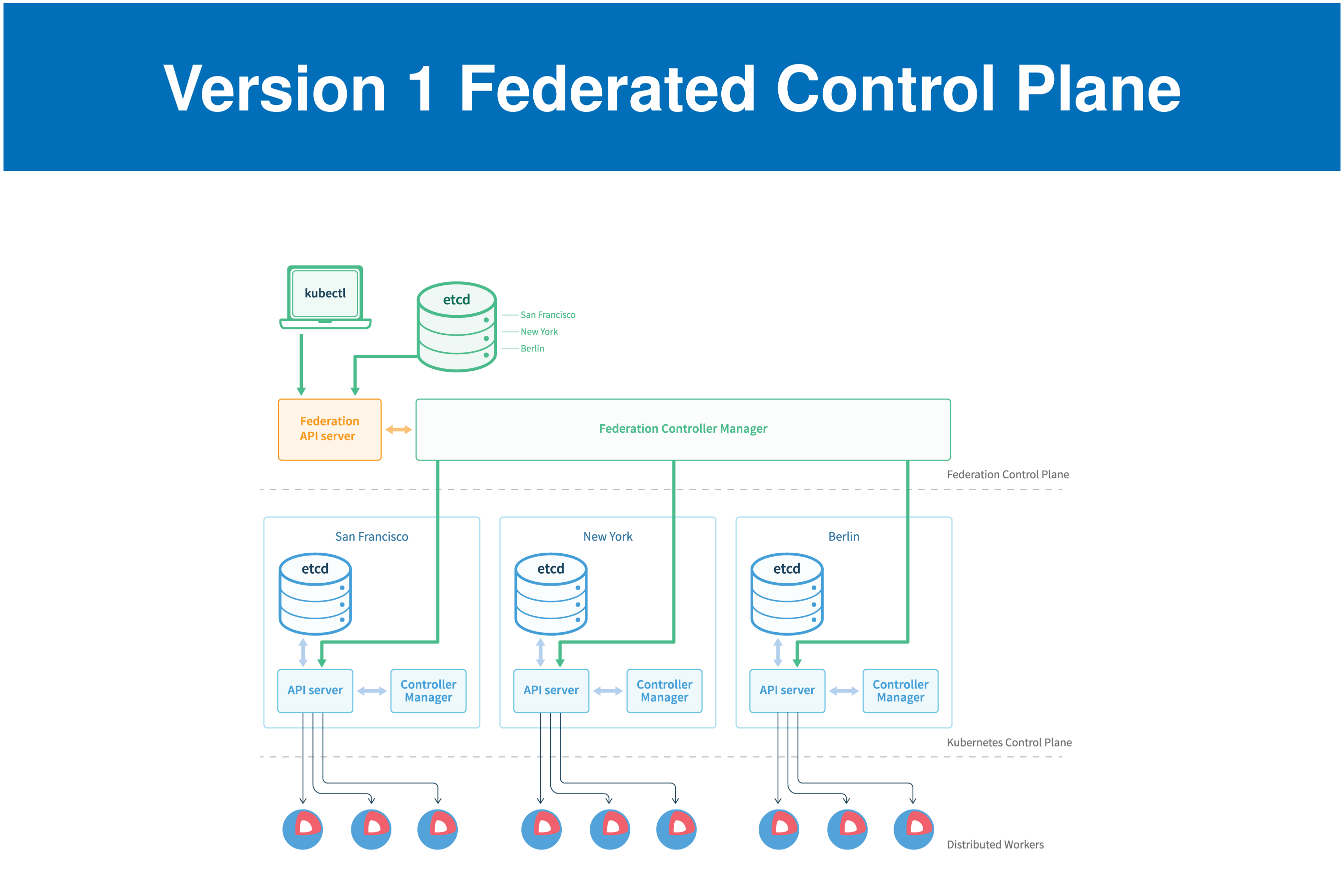

Figure 14.1: **Federation V1 Layout**

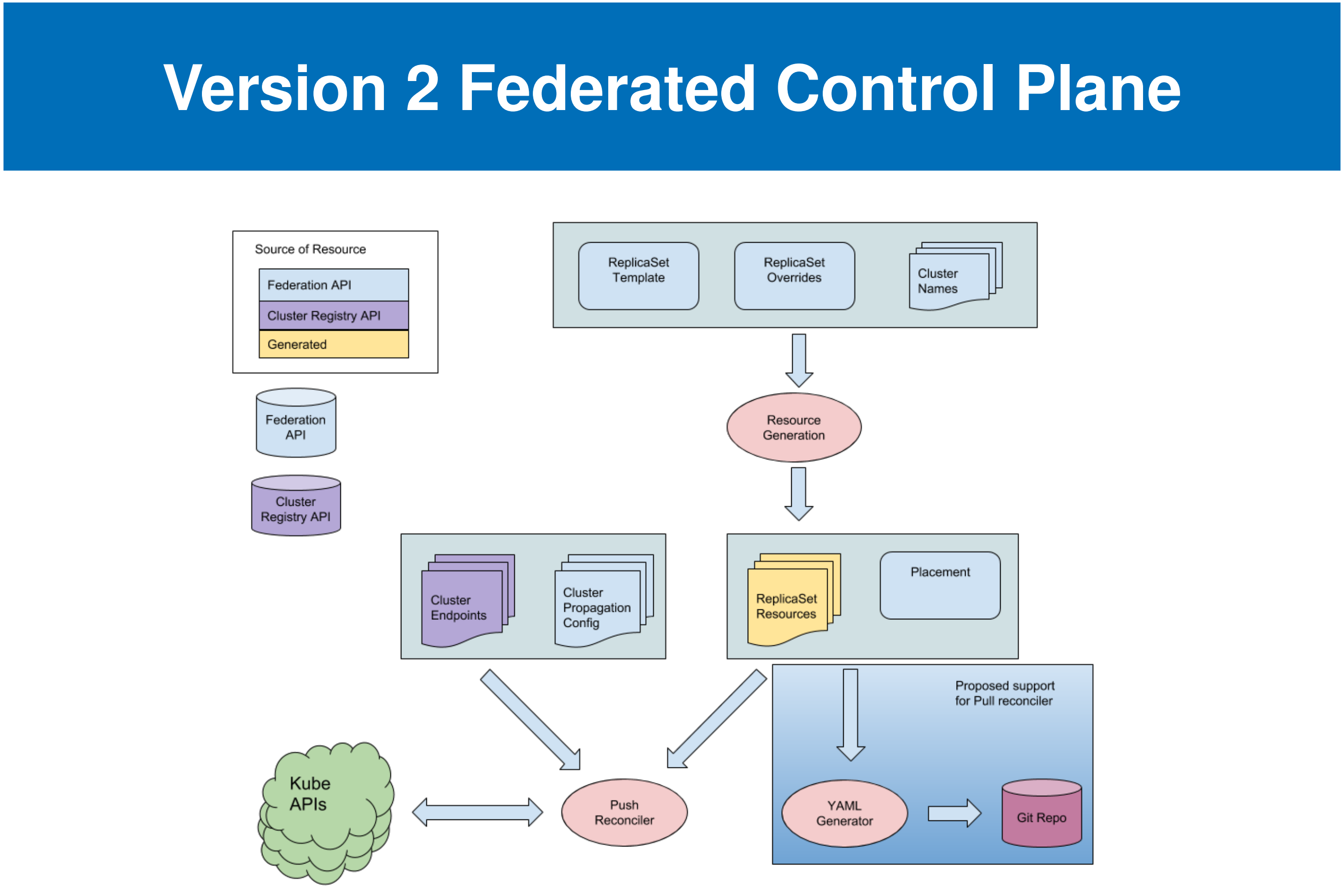

Figure 14.2: **Federation V2 Layout**

Switching Between Clusters

```
$ kubectl config use-context sanfrancisco
$ kubectl get nodes
$ kubectl config use-context chicago
$ kubectl get nodes
$ kubectl --context=paris get nodes
```

Build Federation with kubefed

- Download client tarball
- Choose a host cluster
- Deploy federation control plane
- Add clusters to federation

Using Federated Resources

- Declare Federated API server
- Not all resources are stable
- Replicas spread equally among clusters
- Rolling updates handled using local settings

Federation API Resources

- Cluster
- ConfigMap
- DaemonSet
- Deployment
- Events
- Horizontal Pod Autoscalers (HPA)
- Ingress
- Jobs
- Namespaces
- ReplicaSets
- Secrets

Balancing ReplicaSets

```
annotations:
    federation.kubernetes.io/replica-set-preferences: |
       {
            "rebalance": true,
            "clusters": {
               "new-york": {
                    "minReplicas": 0,
                     "maxReplicas": 20,
                     "weight": 10
                },
                "berlin": {
                    "minReplicas": 1,
                     "maxReplicas": 200,
                     "weight": 20
                  }
              }
        }
```

14.3 Labs

There is no lab to complete for this chapter.

Chapter 15

Helm

15.1 Overview

Deploying Complex Applications

- Package your Kubernetes application via **chart** template
- **Helm** client to request install of **chart**
- **Tiller** creates cluster resources according to **chart**

15.2 Helm

Helm and Tiller

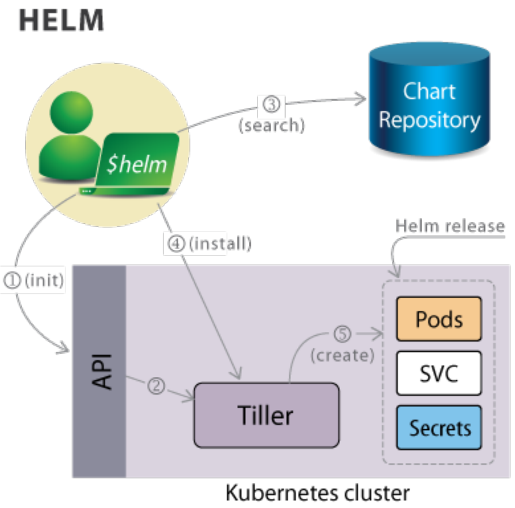

Figure 15.1: **Basic Helm and Tiller Flow**

Chart Contents

```
|-- Chart.yaml
|-- README.md
|-- templates
|   |-- NOTES.txt
|   |-- _helpers.tpl
|   |-- configmap.yaml
|   |-- deployment.yaml
|   |-- pvc.yaml
|   |-- secrets.yaml
|   |-- svc.yaml
|-- values.yaml
```

Templates

```
apiVersion: v1
kind: Secret
metadata:
    name: {{ template "fullname" . }}
    labels:
        app: {{ template "fullname" . }}
        chart: "{{ .Chart.Name }}-{{ .Chart.Version }}"
        release: "{{ .Release.Name }}"
        heritage: "{{ .Release.Service }}"
type: Opaque
data:
   mariadb-root-password: {{ default "" .Values.mariadbRootPassword | b64enc\
                                                                    | quote }}
   mariadb-password: {{ default "" .Values.mariadbPassword | b64enc | quote }}
```

15.3 Using Helm

Initializing Helm

```
$ helm init

...
Tiller (the helm server side component) has been installed into your \
                                        Kubernetes Cluster.
Happy Helming!

$ kubectl get deployments --namespace=kube-system

NAMESPACE     NAME             READY  UP-TO-DATE  AVAILABLE AGE
kube-system   tiller-deploy    1/1    1           1         15s
```

Chart Repositories

- Search for charts
- Add a new repository
- Simple HTTP servers with index file and tarball of charts
- **helm repo**

Deploying a Chart

```
$ helm install testing/redis-standalone

Fetched testing/redis-standalone to redis-standalone-0.0.1.tgz
amber-eel
Last Deployed: Fri Oct 21 12:24:01 2016
Namespace: default
Status: DEPLOYED

Resources:
==> v1/ReplicationController
NAME              DESIRED  CURRENT  READY  AGE
redis-standalone  1        1        0      1s
==> v1/Service
NAME    CLUSTER-IP  EXTERNAL-IP  PORT(S)   AGE
redis   10.0.81.67  <none>       6379/TCP  0s
```


15.4 Labs

Exercise 15.1: Working with Helm and Charts

Overview

helm allows for easy deployment of complex configurations. This could be handy for a vendor to deploy a multi-part application in a single step. Through the use of a `Chart`, or template file, the required components and their relationships are declared. Local agents like **Tiller** use the API to create objects on your behalf. Effectively its orchestration for orchestration.

There are a few ways to install **Helm**. The newest version may require building from source code. We will download a recent, stable version. Once installed we will deploy a `Chart`, which will configure **MariaDB** on our cluster.

Install Helm

1. On the `master` node use **wget** to download the compressed tar file. The short URL below is for: https://storage.googleapis.com/kubernetes-helm/\helm-v2.13.1-linux-amd64.tar.gz

```
student@lfs458-node-1a0a:~$ wget https://bit.ly/2IIn5WW
```

```
<output_omitted>
2IIn5WW             100%[===================>]  21.89M   101MB/s    in 0.2s

2019-04-26 03:43:55 (101 MB/s) - ‘2IIn5WW’ saved [22949819/22949819]
```

2. Uncompress and expand the file.

```
student@lfs458-node-1a0a:~$ tar -xvf 2IIn5WW
```

```
linux-amd64/
linux-amd64/tiller
linux-amd64/README.md
```



```
linux-amd64/helm
linux-amd64/LICENSE
```

3. Copy the **helm** binary to the `/usr/local/bin/` directory, so it is usable via the shell search path.

```
student@lfs458-node-1a0a:~$ sudo cp linux-amd64/helm /usr/local/bin/
```

4. Due to new **RBAC** configuration **helm** is unable to run in the `default` namespace, in this version of Kubernetes. During initialization you could choose to create and declare a new namespace. Other RBAC issues may be encountered even then. In this lab we will create a service account for **tiller**, and give it admin abilities on the cluster. More on **RBAC** in another chapter.

   Begin by creating the `serviceaccount` object.

```
student@lfs458-node-1a0a:~$ kubectl create serviceaccount \
           --namespace kube-system tiller

serviceaccount "tiller" created
```

5. Bind the `serviceaccount` to the admin role called `cluster-admin` inside the `kube-system` namespace.

```
student@lfs458-node-1a0a:~$ kubectl create clusterrolebinding \
          tiller-cluster-rule \
          --clusterrole=cluster-admin \
          --serviceaccount=kube-system:tiller

clusterrolebinding.rbac.authorization.k8s.io/tiller-cluster-rule created
```

6. We can now initialize **helm**. This process will also configure **tiller** the client process. There are several possible options to pass such as `nodeAffinity`, a particular version of software, alternate storage backend, and even a dry-run option to generate JSON or YAML output. The output could be edited and ingested into **kubectl**. We will use default values in this case.

```
student@lfs458-node-1a0a:~$ helm init


<output_omitted>
```

7. Update the `tiller-deploy` deployment to have the service account.

```
student@lfs458-node-1a0a:~$ kubectl -n kube-system patch deployment \
          tiller-deploy -p \
          '{"spec":{"template":{"spec":{"serviceAccount":"tiller"}}}}'
```

```
deployment.extensions/tiller-deploy patched
```

8. Verify the **tiller** pod is running. Examine the logs of the pod. Note that each line of log begins with an tag of the component generating the messages, such as `[main]`, `[storage]`, and `[storage]`.

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

```
<output_omitted>
kube-system   tiller-deploy-84b97f465c-76lvs  1/1  Running 0 30m
```

```
student@lfs458-node-1a0a:~$ kubectl -n kube-system logs \
         tiller-deploy-84b97f465c-76lvs
```

```
<output_omitted>
```

9. View the available sub-commands for **helm**. As with other Kubernetes tools, expect ongoing change.

```
student@lfs458-node-1a0a:~$ helm help
```

```
<output_omitted>
```

10. View the current configuration files, archives and plugins for helm. Return to this directory after you have worked with a `Chart` later in the lab.

```
student@lfs458-node-1a0a:~$ helm home
```

```
/home/student/.helm
```

```
student@lfs458-node-1a0a:~$ ls -R /home/student/.helm/
```

```
/home/student/.helm/:
cache   plugins   repository   starters

/home/student/.helm/cache:
archive
<output_omitted>
```

11. Verify **helm** and **tiller** are responding, also check the current version installed.

```
student@lfs458-node-1a0a:~$ helm version

Client: &version.Version{SemVer:"v2.13.1", GitCommit:"61844....39fbb4", GitTreeState:"clean"}
Server: &version.Version{SemVer:"v2.13.1", GitCommit:"61844....39fbb4", GitTreeState:"clean"}
```

12. Ensure both are upgraded to the most recent stable version.

```
student@lfs458-node-1a0a:~$ helm init --upgrade

$HELM_HOME has been configured at /home/student/.helm.

Tiller (the Helm server-side component) has been upgraded
to the current version.
Happy Helming!
```

13. A Chart is a collection of containers to deploy an application. There is a collection available on https://github.com/kubernetes/charts/tree/master/stable, provided by vendors, or you can make your own. Take a moment and view the current stable Charts. Then search for available stable databases.

```
student@lfs458-node-1a0a:~$ helm search database

NAME                    CHART VERSION   APP VERSION     DESCRIPTION
stable/cockroachdb      2.1.1           2.1.5           CockroachDB is a scalable...
stable/dokuwiki         4.2.0           0.20180422.     DokuWiki is a standards-compliant...
stable/ignite           1.0.0           2.7.0           Apache Ignite is an open-source...
stable/janusgraph       0.2.0           1.0             Open source, scalable graph...
```



```
stable/kubedb                0.1.3          0.8.0-beta.2     DEPRECATED KubeDB by AppsCode...
stable/mariadb               5.11.1         10.1.38          Fast, reliable, scalable,...
<output_omitted>
```

14. We will install the **mariadb**. Take a look at install details https://github.com/kubernetes/charts/tree/master/stable/mariadb#custom-mycnf-configuration The **- -debug** option will create a lot of output. Note the interesting name for the deployment, like `illmannered-salamander`. The output will typically suggest ways to access the software. As well we will indicate that we do not want persistent storage, which would require use to create an available `PV`.

```
student@lfs458-node-1a0a:~$ helm --debug install stable/mariadb \
         --set master.persistence.enabled=false \
         --set slave.persistence.enabled=false


[debug] Created tunnel using local port: '38396'

[debug] SERVER: "localhost:38396"

[debug] Original chart version: ""
[debug] Fetched stable/mariadb to /home/student/.helm/cache/archive/mar...

[debug] CHART PATH: /home/student/.helm/cache/archive/mariadb-5.11.1.tgz

NAME:   illmannered-salamander
<output_omitted>
```

15. Using some of the information at the end of the previous command output we will deploy another container and access the database. We begin by getting the root password for `illmannered-salamander`. Be aware the output lacks a carriage return, so the next prompt will appear on the same line. We will need the password to access the running `MariaDB` database.

```
student@lfs458-node-1a0a:~$ kubectl get secret -n default \
        illmannered-salamander-mariadb \
        -o jsonpath="{.data.mariadb-root-password}" \
        | base64 --decode
```



```
IFBldzAQfx
```

16. Now we will install another container to act as a client for the database. We will use **apt-get** to install client software.

```
student@lfs458-node-1a0a:~$ kubectl run -i --tty ubuntu \
          --image=ubuntu:16.04 --restart=Never -- bash -il
```

Inside container

If you don't see a command prompt, try pressing enter.

```
root@ubuntu:/#

root@ubuntu:/# apt-get update ; apt-get install -y mariadb-client

\textless output\_omitted \textgreater
```

17. Use the client software to access the database. The following command uses the server name and the root password we found in a previous step. Both of yours will be different.

Inside container

```
root@ubuntu:/# mysql -h illmannered-salamander-mariadb -p
Enter password: IFBldzAQfx
Welcome to the MariaDB monitor.  Commands end with ; or \  g.
Your MariaDB connection id is 153
Server version: 10.1.38-MariaDB Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\  h' for help. Type '\  c' to clear the current input statement.
```

```
MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| my_database        |
| mysql              |
| performance_schema |
| test               |
+--------------------+
5 rows in set (0.00 sec)

MariaDB [(none)]>
MariaDB [(none)]> quit
Bye
root@ubuntu:/# exit
```

18. View the `Chart` history on the system. The use of the **-a** option will show all `Charts` including `deleted` and `failed` attempts. The output below shows the current running `Chart` as well as a previously deleted **hadoop** `Chart`. So you can see previous installations.

```
student@lfs458-node-1a0a:~$ helm list -a

NAME                       REVISION UPDATED                     STATUS...
illmannered-salamander     1        Fri Apr 26 03:56:32 2019    DEPLOY...
```

19. Delete the **mariadb** `Chart`. No output should happen from the list.

```
student@lfs458-node-1a0a:~$  helm delete illmannered-salamander

release "illmannered-salamander" deleted
```



```
student@lfs458-node-1a0a:~$ helm list
```

20. Add another repository and view the `Charts` available.

```
student@lfs458-node-1a0a:~$ helm repo add common \
          http://storage.googleapis.com/kubernetes-charts

"common" has been added to your repositories
```

```
student@lfs458-node-1a0a:~$ helm repo list

NAME          URL
stable        https://kubernetes-charts.storage.googleapis.com
local         http://127.0.0.1:8879/charts
common        http://storage.googleapis.com/kubernetes-charts
```

```
student@lfs458-node-1a0a:~$ helm search | less

NAME                            CHART VERSION   APP VERSION   DESCRIPTION
common/acs-engine-autoscaler    2.2.2           2.1.1         DEPRECATED Scales worker...
common/aerospike                0.2.3           v4.5.0.5      A Helm chart for Aerospike in Kubernetes
common/airflow                  2.4.4           1.10.0        Airflow is a platform to ...
<output_omitted>
```

Chapter 16

Security

16.1 Overview

Overview

- Explain how API requests go through the system.
- Compare different types of authentication.
- Configure authorization rules.
- Configure pod policies to control what containers are allowed to do.
- Restrict network traffic using network policies.

16.2 Accessing the API

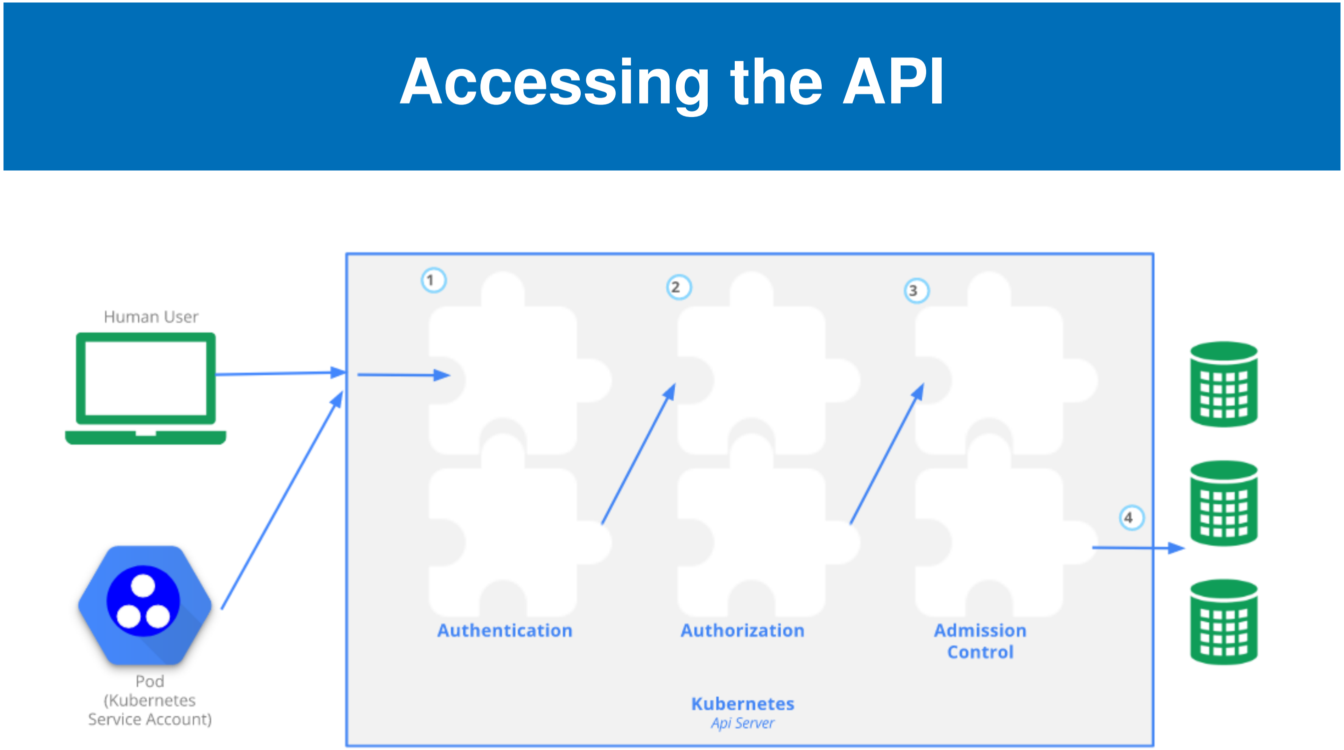

Figure 16.1: **Accessing the API from** `kubernetes.io`

16.3 Authentication and Authorization

Authentication

- One or more **Authenticator Modules** used
  - x509 Client Certs
  - Static Token, Bearer or Bootstrap Token
  - Static Password File
  - Service Account and OpenID Connect Tokens
- Each tried until success, order not guaranteed
- Anonymous access can be enabled. Otherwise 401 response.
- Users are not created by the API, should be managed by an external system.
- Use of `proxy` or `webhook` to LDAP, SAML, Kerberos, alternate x509 etc. instead.
- System accounts are used by processes to access the API

Authorization

- **ABAC**
- **RBAC**
- **WebHook**

ABAC

- Attribute Based Access Control
- Used prior to RBAC

```
{
    "apiVersion": "abac.authorization.kubernetes.io/v1beta1",
    "kind": "Policy",
    "spec": {
        "user": "bob",
        "namespace": "foobar",
        "resource": "pods",
        "readonly": true
    }
}
```

RBAC

- Resources and Operations (verbs)
- Rules
- Roles and ClusterRoles
- Subjects
- RoleBindings and ClusterRoleBindings

RBAC Process Overview

- Determine or create namespace
- Create certificate credentials for user
- Set the credentials for the user to the namespace using a **context**
- Create a role for the expected task set
- Bind the user to the role
- Verify the user has limited access

16.4 Admission Controller

Admission Controller

- Access content of objects created
- Modify or validate content

Security Contexts

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  securityContext:
    runAsNonRoot: true
  containers:
  - image: nginx
    name: nginx
```

16.5 Pod Policies

Pod Security Policies

```
apiVersion: extensions/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restricted
spec:
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  runAsUser:
    rule: MustRunAsNonRoot
  fsGroup:
    rule: RunAsAny
```

16.6 Network Policies

Network Security Policies

- All traffic allowed by default
- Networking `add-ons` must support network policies
- Can be limited to a namespace
- Egress policy type now available
- Can match `PodSelector`, ingress labels and egress labels.

Network Security Policy Example

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: ingress-egress-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
<continued_on_next_slide>
```

Network Security Policy Example Cont.

```
  - namespaceSelector:
      matchLabels:
        project: myproject
  - podSelector:
      matchLabels:
        role: frontend
  ports:
  - protocol: TCP
    port: 6379
egress:
- to:
  - ipBlock:
      cidr: 10.0.0.0/24
  ports:
  - protocol: TCP
    port: 5978
```

Default Policy Example

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

16.7 Labs

Exercise 16.1: Working with TLS

Overview

We have learned that the flow of access to a cluster begins with TLS connectivity, then authentication followed by authorization, finally an admission control plug-in allows advanced features prior to the request being fulfilled. The use of `Initializers` allows the flexibility of a shell-script to dynamically modify the request. As security is an important, ongoing concern, there may be multiple configurations used depending on the needs of the cluster.

Every process making API requests to the cluster must authenticate or be treated as an anonymous user.

While one can have multiple cluster root Certificate Authorities (CA) by default each cluster uses their own, intended for intra-cluster communication. The CA certificate bundle is distributed to each node and as a secret to default service accounts. The **kubelet** is a local agent which ensures local containers are running and healthy.

1. View the **kubelet** on both the master and secondary nodes. The **kube-apiserver** also shows security information such as certificates and authorization mode. As **kubelet** is a **systemd** service we will start looking at that output.

```
student@lfs458-node-1a0a:~$ systemctl status kubelet.service
```

```
  kubelet.service - kubelet: The Kubernetes Node Agent
   Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor preset: en
  Drop-In: /etc/systemd/system/kubelet.service.d
           |__10-kubeadm.conf
<output_omitted>
```

2. If we look at the status output, and follow the cgroup information, which is a long line we where configuration settings are drawn from, we see where the configuration file can be found.

```
 CGroup: /system.slice/kubelet.service
  |--19523 /usr/bin/kubelet .... --config=/var/lib/kubelet/config.yaml ..
```

3. Take a look at the settings in the /var/lib/kubelet/config.yaml file. Among other information we can see the /etc/kubernetes/pki/ directory is used for accessing the **kube-apiserver**. Near the end of the output it also sets the directory to find other pod spec files.

```
student@lfs458-node-1a0a:~$ sudo less /var/lib/kubelet/config.yaml
```

config.yaml

```yaml
address: 0.0.0.0
apiVersion: kubelet.config.k8s.io/v1beta1
authentication:
  anonymous:
    enabled: false
  webhook:
    cacheTTL: 2m0s
    enabled: true
  x509:
    clientCAFile: /etc/kubernetes/pki/ca.crt
```

4. Other agents on the master node interact with the **kube-apiserver**. View the configuration files where these settings are made. This was set in the previous YAML file. Look at one of the files for cert information.

```
student@lfs458-node-1a0a:~$ sudo ls /etc/kubernetes/manifests/

etcd.yaml            kube-controller-manager.yaml
kube-apiserver.yaml  kube-scheduler.yaml
```

```
student@lfs458-node-1a0a:~$ sudo less \
         /etc/kubernetes/manifests/kube-controller-manager.yaml

<output_omitted>
```

5. The use of `tokens` has become central to authorizing component communication. The tokens are kept as **secrets**. Take a look at the current secrets in the `kube-system` namespace.


```
student@lfs458-node-1a0a:~$  kubectl -n kube-system get secrets
NAME                                  TYPE
 DATA       AGE
attachdetach-controller-token-xqr8n  kubernetes.io/service-account-token
 3          5d
bootstrap-signer-token-xbp6s         kubernetes.io/service-account-token
 3          5d
bootstrap-token-i3r13t               bootstrap.kubernetes.io/token
 7          5d
<output_omitted>
```

6. Take a closer look at one of the secrets and the token within. The `certificate-controller-token` could be one to look at. The use of the Tab key can help with long names. Long lines have been truncated in the output below.

```
student@lfs458-node-1a0a:~$ kubectl -n kube-system get secrets \
           certificate<Tab> -o yaml
```

```yaml
apiVersion: v1
data:
  ca.crt: LS0tLS1CRUdJTi.....
  namespace: a3ViZS1zeXN0ZW0=
  token: ZXlKaGJHY2lPaUpTVXpJM....
kind: Secret
metadata:
  annotations:
    kubernetes.io/service-account.name: certificate-controller
    kubernetes.io/service-account.uid: 7dfa2aa0-9376-11e8-8cfb
-42010a800002
  creationTimestamp: 2018-07-29T21:29:36Z
  name: certificate-controller-token-wnrwh
  namespace: kube-system
  resourceVersion: "196"
  selfLink: /api/v1/namespaces/kube-system/secrets/certificate-
```

```yaml
controller-token-wnrwh
  uid: 7dfbb237-9376-11e8-8cfb-42010a800002
type: kubernetes.io/service-account-token
```

7. The **kubectl config** command can also be used to view and update parameters. When making updates this could avoid a typo removing access to the cluster. View the current configuration settings. The keys and certs are redacted from the output automatically.

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

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: REDACTED
<output_omitted>
```

8. View the options, such as setting a password for the admin instead of a key. Read through the examples and options.

```
student@lfs458-node-1a0a:~$ kubectl config set-credentials -h
```

```
Sets a user entry in kubeconfig
<output_omitted>
```

9. Make a copy of your access configuration file. Later steps will update this file and we can view the differences.

```
student@lfs458-node-1a0a:~$ cp ~/.kube/config ~/cluster-api-config
```

10. Explore working with cluster and security configurations both using **kubectl** and **kubeadm**. Among other values, find the name of your cluster. You will need to become `root` to work with **kubeadm**.

```
student@lfs458-node-1a0a:~$ kubectl config <Tab><Tab>
```

```
current-context   get-contexts      set-context       view
delete-cluster    rename-context    set-credentials
delete-context    set               unset
get-clusters      set-cluster       use-context
```

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

root@lfs458-node-1a0a:~# kubeadm token -h

<output_omitted>


root@lfs458-node-1a0a:~# kubeadm config -h

<output_omitted>
```

11. Review the cluster default configuration settings. At over 150 lines there may be some interesting tidbits to the security and infrastructure of the cluster.

```
student@lfs458-node-1a0a:~$  kubeadm config print-default

api:
  advertiseAddress: 10.128.0.2
  bindPort: 6443
  controlPlaneEndpoint: ""
apiVersion: kubeadm.k8s.io/v1alpha2
auditPolicy:
  logDir: /var/log/kubernetes/audit
  logMaxAge: 2
  path: ""
bootstrapTokens:
- groups:
  - system:bootstrappers:kubeadm:default-node-token
  token: abcdef.0123456789abcdef
<output_omitted>
```

Exercise 16.2: Authentication and Authorization

Kubernetes clusters have to types of users `service accounts` and `normal users`, but `normal users` are assumed to be managed by an outside service. There are no objects to represent them and they cannot be added via an API call, but `service accounts` can be added.

We will use **RBAC** to configure access to actions within a `namespace` for a new contractor, `Developer Dan` who will be working on a new project.

1. Create two namespaces, one for `production` and the other for `development`.

```
student@lfs458-node-1a0a:~$ kubectl create ns development
namespace "development" created

student@lfs458-node-1a0a:~$ kubectl create ns production
namespace "production" created
```

2. View the current clusters and context available. The context allows you to configure the cluster to use, namespace and user for **kubectl** commands in an easy and consistent manner.

```
student@lfs458-node-1a0a:~$ kubectl config get-contexts
CURRENT   NAME                          CLUSTER      AUTHINFO           NAMESPACE
*         kubernetes-admin@kubernetes   kubernetes   kubernetes-admin
```

3. Create a new user `DevDan` and assign a password of `lfs458`.

```
student@lfs458-node-1a0a:~$ sudo useradd -s /bin/bash DevDan

student@lfs458-node-1a0a:~$ sudo passwd DevDan
Enter new UNIX password:  lfs458
Retype new UNIX password:  lfs458
passwd: password updated successfully
```

4. Generate a private key then Certificate Signing Request (CSR) for `DevDan`.


```
student@lfs458-node-1a0a:~$ openssl genrsa -out DevDan.key 2048

Generating RSA private key, 2048 bit long modulus
......+++
.........+++
e is 65537 (0x10001)


student@lfs458-node-1a0a:~$ openssl req -new -key DevDan.key \
      -out DevDan.csr -subj "/CN=DevDan/O=development"
```

5. Using 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@lfs458-node-1a0a:~$ sudo openssl x509 -req -in DevDan.csr \
     -CA /etc/kubernetes/pki/ca.crt \
     -CAkey /etc/kubernetes/pki/ca.key \
     -CAcreateserial \
     -out DevDan.crt -days 45

Signature ok
subject=/CN=DevDan/O=development
Getting CA Private Key
```

6. Update the access config file to reference the new key and certificate. Normally we would move them to a safe directory instead of a non-root user's home.

```
student@lfs458-node-1a0a:~$ kubectl config set-credentials DevDan \
      --client-certificate=/home/student/DevDan.crt \
      --client-key=/home/student/DevDan.key

User "DevDan" set.
```

7. View the update to your credentials file. Use **diff** to compare against the copy we made earlier.

```
student@lfs458-node-1a0a:~$ diff cluster-api-config .kube/config
```



```
9a10,14
>       namespace: development
>       user: DevDan
>     name: DevDan-context
> - context:
>       cluster: kubernetes
15a21,25
> - name: DevDan
>     user:
>       as-user-extra: {}
>       client-certificate: /home/student/DevDan.crt
>       client-key: /home/student/DevDan.key
```

8. We will now create a context. For this we will need the name of the cluster, namespace and CN of the user we set or saw in previous steps.

```
student@lfs458-node-1a0a:~$ kubectl config set-context DevDan-context \
       --cluster=kubernetes \
       --namespace=development \
       --user=DevDan

Context "DevDan-context" created.
```

9. Attempt to view the Pods inside the DevDan-context. Be aware you will get an error.

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

Error from server (Forbidden): pods is forbidden: User "DevDan"
cannot list pods in the namespace "development"
```

10. Verify the context has been properly set.

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

CURRENT   NAME                 CLUSTER      AUTHINFO          NAMESPACE
          DevDan-context       kubernetes   DevDan            development
*         kubernetes-admin@kubernetes kubernetes kubernetes-admin
```


11. Again check the recent changes to the cluster access config file.

```
student@lfs458-node-1a0a:~$ diff cluster-api-config .kube/config
<output_omitted>
```

12. We will now create a YAML file to associate RBAC rights to a particular namespace and `Role`.

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

role-dev.yaml

```yaml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
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@lfs458-node-1a0a:~$ kubectl create -f role-dev.yaml
role.rbac.authorization.k8s.io/developer created
```

14. Now we create a `RoleBinding` to associate the `Role` we just created with a user. Create the object when the file has been created.

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

rolebind.yaml

```yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: developer-role-binding
  namespace: development
subjects:
- kind: User
  name: DevDan
  apiGroup: ""
roleRef:
  kind: Role
  name: developer
  apiGroup: ""
```

```
student@lfs458-node-1a0a:~$ kubectl apply -f rolebind.yaml
rolebinding.rbac.authorization.k8s.io/developer-role-binding created
```

15. Test the context again. This time it should work. There are no `Pods` running so you should get a response of `No resources found`.

```
student@lfs458-node-1a0a:~$  kubectl --context=DevDan-context get pods
No resources found.
```

16. Create a new pod, verify it exists, then delete it.

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

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

```
NAME                     READY   STATUS            RESTARTS   AGE
nginx-7c87f569d-7gb9k    1/1     Running           0          5s
```

```
student@lfs458-node-1a0a:~$ kubectl --context=DevDan-context delete \
     deploy nginx
```

```
deployment.extensions "nginx" deleted
```

17. We will now create a different context for production systems. The `Role` will only have the ability to view, but not create or delete resources. Begin by copying and editing the `Role` and `RoleBindings` YAML files.

```
student@lfs458-node-1a0a:~$ cp role-dev.yaml role-prod.yaml
```

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

role-prod.yaml

```yaml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
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@lfs458-node-1a0a:~$ cp rolebind.yaml rolebindprod.yaml
```

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

rolebindprod.yaml

```yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: production-role-binding
  namespace: production
subjects:
- kind: User
  name: DevDan
  apiGroup: ""
roleRef:
  kind: Role
  name: dev-prod
  apiGroup: ""
```

18. Create both new objects.

```
student@lfs458-node-1a0a:~$ kubectl apply -f role-prod.yaml
role.rbac.authorization.k8s.io/dev-prod created

student@lfs458-node-1a0a:~$ kubectl apply -f rolebindprod.yaml
rolebinding.rbac.authorization.k8s.io/production-role-binding created
```

19. Create the new context for production use.

```
student@lfs458-node-1a0a:~$ kubectl config set-context ProdDan-context \
       --cluster=kubernetes \
       --namespace=production \
       --user=DevDan
Context "ProdDan-context" created.
```

20. Verify that user DevDan can view pods using the new context.

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

```
No resources found.
```

21. Try to create a Pod in production. The developer should be Forbidden.

```
student@lfs458-node-1a0a:~$ kubectl --context=ProdDan-context create \
      deployment nginx --image=nginx
```

```
Error from server (Forbidden): deployments.extensions is forbidden:
User "DevDan" cannot create deployments.extensions in the
namespace "production"
```

22. View the details of a role.

```
student@lfs458-node-1a0a:~$ kubectl -n production describe role dev-prod
```

```
Name:         dev-prod
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration=
{"apiVersion":"rbac.authorization.k8s.io/v1beta1","kind":"Role"
,"metadata":{"annotations":{},"name":"dev-prod","namespace":
"production"},"rules":[{"api...
PolicyRule:
  Resources         Non-Resource URLs  Resource Names  Verbs
  ---------         -----------------  --------------  -----
  deployments       []                 []              [get list watch]
  deployments.apps  []                 []              [get list watch]
<output_omitted>
```

23. Experiment with other subcommands in both contexts. They should match those listed in the respective roles.

Exercise 16.3: Admission Controllers

The last stop before a request is sent to the API server is an `admission control plug-in`. They interact with features such as setting parameters like a default storage class, checking resource quotas, or security settings. A newer feature (v1.7.x) is dynamic controllers which allow new controllers to be ingested or configured at runtime.

1. View the current `admission controller` settings. Unlike earlier versions of Kubernetes the controllers are now compiled into the server, instead of being passed at run-time. Instead of a list of which controllers to use we can enable and disable specific plugins.

```
student@lfs458-node-1a0a:~$ sudo grep admission \
    /etc/kubernetes/manifests/kube-apiserver.yaml

    - --disable-admission-plugins=PersistentVolumeLabel
    - --enable-admission-plugins=NodeRestriction
```


THE LINUX FOUNDATION