© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

LFD259

Kubernetes for Developers

Version 2020-10-15

Ghazela Technology
Academy

Copyright © 2020   adresse : 23, rue el Yakoute ,cité la Ghazelle ,Ariana
MF :1469595/X/A/M/000 – RC : B01912050
Contact : contact at ghazelatc.com
Téléphone :+216 71 866 142 , +216 27 862 155, +216 54 828 018

THE LINUX FOUNDATION

Version 2020-10-15

© Copyright the Linux Foundation 2020. All rights reserved.

© Copyright the Linux Foundation 2020. All rights reserved.

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

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

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

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

Published by:

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

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

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

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

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

Ghazela Technology
Academy

Copyright © 2020 adresse : 23, rue el Yakoute ,cité la Ghazelle ,Ariana
MF :1469595/X/A/M/000 – RC : B01912050
Contact : contact at ghazelatc.com
Téléphone :+216 71 866 142 , +216 27 862 155, +216 54 828 018

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

Contents

Ghazela Technology
Academy
Copyright © 2020 adresse : 23, rue el Yakoute ,cité la Ghazelle ,Ariana
MF :1469595/X/A/M/000 – RC : B01912050
Contact : contact at ghazelatc.com
Téléphone :+216 71 866 142 , +216 27 862 155, +216 54 828 018

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

Ghazela Technology
Academy

Copyright © 2020 adresse : 23, rue el Yakoute ,cité la Ghazelle ,Ariana
MF :1469595/X/A/M/000 – RC : B01912050
Contact : contact at ghazelatc.com
Téléphone :+216 71 866 142 , +216 27 862 155, +216 54 828 018

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

Chapter 2

Kubernetes Architecture

Exercise 2.1: Overview and Preliminaries

We will create a two-node **Ubuntu 18.04** cluster. Using two nodes allows an understanding of some issues and configurations found in a production environment. Currently 2 vCPU and 8G of memory allows for quick labs. Other Linux distributions should work in a very similar manner, but have not been tested.

Very Important

Regardless of the platform used (**VirtualBox**, **VMWare**, **AWS**, **GCE** or even bare metal) please remember that security software like **SELinux**, **AppArmor**, and firewall configurations can prevent the labs from working. While not something to do in production consider disabling the firewall and security software.

GCE requires a new `VPC` to be created and a rule allowing all traffic to be included. The use of **wireshark** can be a helpful place to start with troubleshooting network and connectivity issues if you're unable to open all ports.

The **kubeadm** utility currently requires that swap be turned off on every node. The **swapoff -a** command will do this until the next reboot, with various methods to disable swap persistently. Cloud providers typically deploy instances with swap disabled.

Download shell scripts and YAML files

To assist with setting up your cluster please download the tarball of shell scripts and YAML files. The `k8sMaster.sh` and `k8sSecond.sh` scripts deploy a Kubernetes cluster using **kubeadm** and use `Project Calico` for networking. Should the file not be found you can always use a browser to investigate the parent directory.

```
$ wget https://training.linuxfoundation.org/cm/LFD259/LFD259_V2020-10-15_SOLUTIONS.tar.bz2 \
              --user=LFtraining --password=Penguin2014

$ tar -xvf LFD259_V2020-10-15_SOLUTIONS.tar.bz2
```

(Note: depending on your software, 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!)

Exercise 2.2: Deploy a New Cluster

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

Deploy a Master Node using Kubeadm

1. Log into your nodes using **PuTTY** or using **SSH** from a terminal window. Unless the instructor tells you otherwise the user name to use will be **student**. You may need to change the permissions on the `pem` or `ppk` file as shown in the following commands. Your file and node IP address will probably be different.

```
localTerm:~$ chmod 400 LFD459.pem
localTerm:~$ ssh -i LFD459.pem student@WW.XX.YY.ZZ
```

```
student@ckad-1:~$
```

2. Review the script to install and begin the configuration of the master kubernetes server. You may need to change the **find** command search directory which uses tilde for your home directory depending on how and where you downloaded the tarball.

> A **find** command is shown if you want to locate and copy to the current directory instead of creating the file. Mark the command for reference as it may not be shown for future commands.
>
> ```
> student@ckad-1:~$ find $HOME -name <YAML File>
> student@ckad-1:~$ cp LFD259/<Some Path>/<YAML File> .
> ```

```
student@ckad-1:~$ find $HOME -name k8sMaster.sh
```

```
student@ckad-1:~$ more LFD259/SOLUTIONS/s_02/k8sMaster.sh
```

SH k8sMaster.sh

```
#!/bin/bash -x
## TxS 08-2020
## v1.19.0 CKAD
echo "This script is written to work with Ubuntu 18.04"
sleep 3
echo
echo "Disable swap until next reboot"
echo
sudo swapoff -a

echo "Update the local node"
sudo apt-get update && sudo apt-get upgrade -y
echo
echo "Install Docker"
sleep 3

sudo apt-get install -y docker.io
echo
echo "Install kubeadm, kubelet, and kubectl"
sleep 3

sudo sh -c
↪  "echo 'deb http://apt.kubernetes.io/ kubernetes-xenial main' >> /etc/apt/sources.list.d/kubernetes.list"

sudo sh -c "curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -"

sudo apt-get update

sudo apt-get install -y kubeadm=1.19.0-00 kubelet=1.19.0-00 kubectl=1.19.0-00

sudo apt-mark hold kubelet kubeadm kubectl
```

© Copyright the Linux Foundation 2020. All rights reserved.

```sh
echo
echo "Installed - now to get Calico Project network plugin"

## If you are going to use a different plugin you'll want
## to use a different IP address, found in that plugins
## readme file.

sleep 3

sudo kubeadm init --kubernetes-version 1.19.0 --pod-network-cidr 192.168.0.0/16

sleep 5

echo "Running the steps explained at the end of the init output for you"

mkdir -p $HOME/.kube

sleep 2

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sleep 2

sudo chown $(id -u):$(id -g) $HOME/.kube/config

echo "Apply Calico network plugin from ProjectCalico.org"
echo "If you see an error they may have updated the yaml file"
echo "Use a browser, navigate to the site and find the updated file"

kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml

echo
echo
sleep 3
echo "You should see this node in the output below"
echo "It can take up to a mintue for node to show Ready status"
echo
kubectl get node
echo
echo
echo "Script finished. Move to the next step"
```

3. Run the script as an argument to the **bash** shell. You will need the `kubeadm join` command shown near the end of the output when you add the worker/minion node in a future step. Use the **tee** command to save the output of the script, in case you cannot scroll back to find the `kubeadm join` in the script output. Please note the following is one command and then its output.

   Using **Ubuntu 18** you will be asked questions during the installation. Allow restarts and use the local, installed software if asked during the update, usually option 2.

   Copy files to your home directory first.

```
student@ckad-1:~$ cp LFD259/SOLUTIONS/s_02/k8sMaster.sh .

student@ckad-1:~$ bash k8sMaster.sh | tee $HOME/master.out
```

```
<output_omitted>

Your Kubernetes master has initialized successfully!

      To start using your cluster, you need to run the
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
   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 \verb?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 10.128.0.3:6443 --token 69rdjq.2x20l2j9ncexy37b
 --discovery-token-ca-cert-hash

sha256:72143e996ef78301191b9a42184124416aebcf0c7f363adf9208f9fa599079bd

<output_omitted>

+ kubectl get node
NAME              STATUS     ROLES     AGE       VERSION
ckad-1            NotReady   master    18s       v1.19.0
+ echo
+ echo 'Script finished. Move to the next step'
Script finished. Move to the next step
```

Deploy a Minion Node

4. Open a separate terminal into your **second node**. Having both terminal sessions allows you to monitor the status of the cluster while adding the second node. Change the color or other characteristic of the second terminal to make it visually distinct from the first. This will keep you from running commands on the incorrect instance, which probably won't work.

   Use the previous **wget** command to download the tarball to the second node. Extract the files with **tar** as before. Find and copy the `k8sSecond.sh` file to `student`'s home directory then view it. You should see the same early steps as on the master system.

```
student@ckad-2:~$ more k8sSecond.sh
```

SH k8sSecond.sh

```
#!/bin/bash -x
## TxS 08-2020
## CKAD for 1.19.0
##
echo "  This script is written to work with Ubuntu 18.04"
echo
sleep 3
echo "  Disable swap until next reboot"
echo
sudo swapoff -a

echo "  Update the local node"
sleep 2
sudo apt-get update && sudo apt-get upgrade -y
echo
sleep 2
```

© Copyright the Linux Foundation 2020. All rights reserved.

```
echo "  Install Docker"
sleep 3
sudo apt-get install -y docker.io

echo
echo "  Install kubeadm, kubelet, and kubectl"
sleep 2
sudo sh -c
↪  "echo 'deb http://apt.kubernetes.io/ kubernetes-xenial main' >> /etc/apt/sources.list.d/kubernetes.list"

sudo sh -c "curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -"

sudo apt-get update

sudo apt-get install -y kubeadm=1.19.0-00 kubelet=1.19.0-00 kubectl=1.19.0-00

sudo apt-mark hold kubelet kubeadm kubectl
echo
echo "  Script finished. You now need the kubeadm join command"
echo "  from the output on the master node"
echo
```

5. Run the script on the **second node**. Again please note you may have questions during the update. Allow daemons to restart and use the local installed version, usually option 2.

```
student@ckad-2:˜$ bash k8sSecond.sh
```

```
<output_omitted>
```

6. When the script is done the minion node is ready to join the cluster. The `kubeadm join` statement can be found near the end of the `kubeadm init` output on the master node. It should also be in the file `master.out` as well. Your nodes will use a different IP address and hashes than the example below. You'll need to pre-pend **sudo** to run the script copied from the master node. Also note that some non-Linux operating systems and tools insert extra characters when multi-line samples are copied and pasted. Copying one line at a time solves this issue.

```
student@ckad-2:˜$ sudo kubeadm join --token 118c3e.83b49999dc5dc034 \
 10.128.0.3:6443 --discovery-token-ca-cert-hash \
sha256:40aa946e3f53e38271bae24723866f56c86d77efb49aedeb8a70cc189bfe2e1d
```

```
<output_omitted>
```

Configure the Master Node

7. Return to the master node. Install a text editor. While the lab uses **vim**, any text editor such as **emacs** or **nano** will work. Be aware that Windows editors may have issues with special characters. Also install the **bash-completion** package, if not already installed. Use the locally installed version of a package if asked.

```
student@ckad-1:˜$ sudo apt-get install bash-completion vim -y
```

```
<output_omitted>
```

8. We will configure command line completion and verify both nodes have been added to the cluster. The first command will configure completion in the current shell. The second command will ensure future shells have completion. You may need to exit the shell and log back in for command completion to work without error.

```
student@ckad-1:˜$ source <(kubectl completion bash)

student@ckad-1:˜$ echo "source <(kubectl completion bash)" >> $HOME/.bashrc
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

9. Verify that both nodes are part of the cluster. And show a `Ready` state.

```
student@ckad-1:~$ kubectl get node
```

```
NAME        STATUS     ROLES     AGE         VERSION
ckad-1      NotReady   master    4m11s       v1.19.1
ckad-2      NotReady   <none>    3m6s        v1.19.1
```

10. We will use the **kubectl** command for the majority of work with Kubernetes. Review the help output to become familiar with commands options and arguments.

```
student@ckad-1:~$ kubectl --help
```

```
kubectl controls the Kubernetes cluster manager.

Find more information at:
  https://kubernetes.io/docs/reference/kubectl/overview/

Basic Commands (Beginner):
  create         Create a resource from a file or from stdin.
  expose         Take a replication controller, service,
 deployment or pod and expose it as a new Kubernetes Service
  run            Run a particular image on the cluster
  set            Set specific features on objects

Basic Commands (Intermediate):
<output_omitted>
```

11. With more than 40 arguments, you can explore each also using the `--help` option. Take a closer look at a few, starting with `taint` for example.

```
student@ckad-1:~$ kubectl taint --help
```

```
Update the taints on one or more nodes.

  * A taint consists of a key, value, and effect. As an argument
   here, it is expressed as key=value:effect.
  * The key must begin with a letter or number, and may contain
   letters, numbers, hyphens, dots, and underscores, up to
   253 characters.
  * Optionally, the key can begin with a DNS subdomain prefix
   and a single '/',
like example.com/my-app
<output_omitted>
```

12. By default the master node will not allow general containers to be deployed for security reasons. This is via a `taint`. Only containers which tolerate this taint will be scheduled on this node. As we only have two nodes in our cluster we will remove the taint, allowing containers to be deployed on both nodes. This is not typically done in a production environment for security and resource contention reasons. The following command will remove the taint from all nodes, so you should see one success and one `not found` error. The worker/minion node does not have the taint to begin with. Note the **minus sign** at the end of the command, which removes the preceding value.

```
student@ckad-1:~$  kubectl describe nodes | grep -i Taint
```

```
Taints:             node-role.kubernetes.io/master:NoSchedule
Taints:             <node>
```

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

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

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

13. Check that both nodes are without a `Taint`. If they both are without taint the `nodes` should now show as `Ready`. It may take a minute or two for all infrastructure pods to enter `Ready` state, such that the `nodes` will show a `Ready` state.

```
student@ckad-1:~$ kubectl describe nodes | grep -i taint
```

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

```
student@ckad-1:~$ kubectl get nodes
```

```
NAME                 STATUS   ROLES    AGE     VERSION
ckad-1               Ready    master   6m1s    v1.19.0
ckad-2               Ready    <none>   5m31s   v1.19.0
```

Exercise 2.3: Create a Basic Pod

1. The smallest unit we directly control with Kubernetes is the pod. We will create a pod by creating a minimal YAML file. First we will get a list of current API objects and their `APIGROUP`. If value is not shown it may not exist, as with `SHORTNAMES`. Note that `pods` does not declare an `APIGROUP`. At the moment this indicates it is part of the stable `v1` group.

```
student@ckad-1:~$ kubectl api-resources
```

```
NAME              SHORTNAMES  APIGROUP   NAMESPACED   KIND
bindings                                 true         Binding
componentstatuses cs                     false        ComponentStatus
configmaps        cm                     true         ConfigMap
endpoints         ep                     true         Endpoints
.....
pods              po                     true         Pod
....
```

2. Finding no declared `APIGROUP` we will use `v1` to denote a stable object. With that information we will add the other three required sections such as `metadata`, with a `name`, and `spec` which declares which **Docker** image to use and a `name` for the container. We will create an eight line YAML file. White space and indentation matters. Don't use **Tab**s. There is a `basic.yaml` file available in the tarball, as well as `basic-later.yaml` which shows what the file will become and can be helpful for figuring out indentation.

basic.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: basicpod
spec:
  containers:
  - name: webcont
    image: nginx
```

3. Create the new pod using the recently created YAML file.

```
student@ckad-1:~$ kubectl create -f basic.yaml
```

```
pod/basicpod created
```

4. Make sure the pod has been created then use the **describe** sub-command to view the details. Among other values in the output you should be about to find the image and the container name.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
student@ckad-1:~$ kubectl get pod
```

```
NAME      READY   STATUS    RESTARTS   AGE
basicpod  1/1     Running   0          23s
```

```
student@ckad-1:~$ kubectl describe pod basicpod
```

```
Name:                basicpod
Namespace:           default
Priority:            0
<output_omitted>
```

5. Shut down the pod and verify it is no longer running.

```
student@ckad-1:~$ kubectl delete pod basicpod
```

```
pod "basicpod" deleted
```

```
student@ckad-1:~$ kubectl get pod
```

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

6. We will now configure the pod to expose port 80. This configuration does not interact with the container to determine what port to open. We have to know what port the process inside the container is using, in this case port 80 as a web server. Add two lines to the end of the file. Line up the indentation with the image declaration.

```
student@ckad-1:~$ vim basic.yaml
```

basic.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: basicpod
spec:
  containers:
  - name: webcont
    image: nginx
    ports:                        #<--Add this and following line
    - containerPort: 80
```

7. Create the pod and verify it is running. Use the `-o wide` option to see the internal IP assigned to the pod, as well as `NOMINATED NODE`, which is used by the scheduler and `READINESS GATES` which show if experimental features are enabled. Using **curl** and the pods IP address you should get the default nginx welcome web page.

```
student@ckad-1:~$ kubectl create -f basic.yaml
```

```
pod/basicpod created
```

```
student@ckad-1:~$ kubectl get pod -o wide
```

```
NAME     READY STATUS  RESTARTS AGE  IP           NODE
  NOMINATED NODE  READINESS GATES
basicpod 1/1   Running 0        9s   192.168.1.3  ckad-1
  <none>          <none>
```

```
student@ckad-1:~$ curl http://192.168.1.3
```

© Copyright the Linux Foundation 2020. All rights reserved.

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

<output_omitted>
```

```
student@ckad-1:~$ kubectl delete pod basicpod
```

```
pod "basicpod" deleted
```

8. We will now create a simple service to expose the pod to other nodes and pods in the cluster. The `service` YAML will have the same four sections as a pod, but different `spec` configuration and the addition of a `selector`.

```
student@ckad-1:~$ vim basicservice.yaml
```

basicservice.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
    name: basicservice
spec:
  selector:
    type: webserver
  ports:
  - protocol: TCP
    port: 80
```

9. We will also add a `label` to the pod and a `selector` to the service so it knows which object to communicate with.

```
student@ckad-1:~$ vim basic.yaml
```

basic.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: basicpod
  labels:                   #<-- Add this line
    type: webserver         #<-- and this line which matches selector
spec:
....
```

10. Create the new pod and service. Verify both have been created.

```
student@ckad-1:~$ kubectl create -f basic.yaml
```

```
pod/basicpod created
```

```
student@ckad-1:~$ kubectl create -f basicservice.yaml
```

```
service/basicservice created
```

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
student@ckad-1:~$ kubectl get pod
```

```
NAME       READY   STATUS    RESTARTS   AGE
basicpod   1/1     Running   0          110s
```

```
student@ckad-1:~$ kubectl get svc
```

```
NAME          TYPE        CLUSTER-IP     EXTERNAL-IP  PORT(S)    AGE
basicservice  ClusterIP   10.96.112.50   <none>       80/TCP     14s
kubernetes    ClusterIP   10.96.0.1      <none>       443/TCP    4h
```

11. Test access to the web server using the `CLUSTER-IP` for the `basicservice`.

```
student@ckad-1:~$ curl http://10.96.112.50
```

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

<output_omitted>
```

12. We will now expose the service to outside the cluster as well. Delete the service, edit the file and add a `type` declaration.

```
student@ckad-1:~$ kubectl delete svc basicservice
```

```
service "basicservice" deleted
```

```
student@ckad-1:~$  vim basicservice.yaml
```

basicservice.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
    name: basicservice
spec:
  selector:
    type: webserver
  type: NodePort       #<--Add this line
  ports:
  - protocol: TCP
    port: 80
```

13. Create the service again. Note there is a different `TYPE` and `CLUSTER-IP` and also a high-numbered port.

```
student@ckad-1:~$  kubectl create -f basicservice.yaml
```

```
service/basicservice created
```

```
student@ckad-1:~$ kubectl get svc
```

```
NAME         TYPE      CLUSTER-IP     EXTERNAL-IP PORT(S)        AGE
basicservice NodePort  10.100.139.155 <none>      80:31514/TCP   3s
kubernetes   ClusterIP 10.96.0.1      <none>      443/TCP        47h
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

14. Using the public IP address of the node and the high port you should be able to test access to the webserver. In the example below the public IP is `35.238.3.83`, yours will be different. The high port will also probably be different. Note that testing from within a GCE or AWS node will not work. Use a local to you terminal or web browser to test.

```
local$ curl http://35.238.3.83:31514
```

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

Exercise 2.4: Multi-Container Pods

Using a single container per pod allows for the most granularity and decoupling. There are still some reasons to deploy multiple containers, sometimes called `composite containers`, in a single pod. The secondary containers can handle logging or enhance the primary, the `sidecar` concept, or acting as a proxy to the outside, the `ambassador` concept, or modifying data to meet an external format such as an `adapter`. All three concepts are secondary containers to perform a function the primary container does not.

1. We will add a second container to the pod to handle logging. Without going into details of how to use **fluentd** we will add a logging container to the exiting pod from its own repository. The second container would act as a `sidecar`. At this state we will just add the second container and verify it is running. In the **Deployment Configuration** chapter we will continue to work on this pod by adding persistent storage and configure **fluentd** via a `configMap`.

   Edit the YAML file and add a **fluentd** container. The dash should line up with the previous container dash. At this point a name and image should be enough to start the second container.

```
student@ckad-1:~$ vim basic.yaml
```

basic.yaml

```yaml
....
  containers:
  - name: webcont
    image: nginx
    ports:
    - containerPort: 80
  - name: fdlogger
    image: fluent/fluentd
```

2. Delete and create the pod again. The commands can be typed on a single line, separated by a semicolon. This time you should see `2/2` under the `READY` column. You should also find information on the **fluentd** container inside of the `kubectl describe` output.

```
student@ckad-1:~$ kubectl delete pod basicpod ; kubectl create -f basic.yaml
```

```
pod "basicpod" deleted
pod/basicpod created
```

```
student@ckad-1:~$ kubectl get pod
```

```
NAME       READY   STATUS    RESTARTS   AGE
basicpod   2/2     Running   0          2m8s
```

Ghazela Technology Academy

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
student@ckad-1:~$ kubectl describe pod basicpod
```

```
Name:               basicpod
Namespace:          default
Priority:           0
Node:               ckad-1/10.128.0.11

....

  fdlogger:
    Container ID:   docker://f0649457217f00175ce9aec35022d0b238b9b....
    Image:          fluent/fluentd

....
```

3. For now shut down the pod. We will use it again in a future exercise.

```
student@ckad-1:~$ kubectl delete pod basicpod
```

```
pod "basicpod" deleted
```

Exercise 2.5: Create a Simple Deployment

Creating a pod does not take advantage of orchestration abilities of Kubernetes. We will now create a Deployment which gives us scalability, reliability, and updates.

1. Now run a containerized webserver **nginx**. Use **kubectl create** to create a simple, single replica deployment running the nginx web server. It will create a single pod as we did previously but with new controllers to ensure it runs as well as other features.

```
student@ckad-1:~$  kubectl create deployment firstpod --image=nginx
```

```
deployment.apps/firstpod created
```

2. Verify the new deployment exists and the desired number of pods matches the current number. Using a comma, you can request two resource types at once. The **Tab** key can be helpful. Type enough of the word to be unique and press the **Tab** key, it should complete the word. The deployment should show a number 1 for each value, such that the desired number of pods matches the up-to-date and running number. The pod should show zero restarts.

```
student@ckad-1:~$ kubectl get deployment,pod
```

```
NAME                       READY UP-TO-DATE AVAILABLE AGE
deployment.apps/firstpod   1/1   1          1         2m42s

NAME                           READY STATUS  RESTARTS AGE
pod/firstpod-7d88d7b6cf-lrsbk  1/1   Running 0        2m42s
```

3. View the details of the deployment, then the pod. Work through the output slowly. Knowing what a healthy deployment and looks like can be helpful when troubleshooting issues. Again the **Tab** key can be helpful when using long auto-generated object names. You should be able to type firstpod**Tab** and the name will complete when viewing the pod.

```
student@ckad-1:~$ kubectl describe deployment firstpod
```

© Copyright the Linux Foundation 2020. All rights reserved.

```
Name:                 firstpod
Namespace:            default
CreationTimestamp:    Wed, 15 Arp 2020 17:17:25 +0000
Labels:               app=firstpod
Annotations:          deployment.kubernetes.io/revision=1
Selector:             app=firstpod
Replicas:             1 desired | 1 updated | 1 total | 1 available....
StrategyType:         RollingUpdate
MinReadySeconds:      0
<output_omitted>
```

```
student@ckad-1:~$ kubectl describe pod firstpod-6bb4574d94-rqk76
```

```
Name:               firstpod-6bb4574d94-rqk76
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:               ckad-1/10.128.0.2
Start Time:         Wed, 15 Apr 2020 17:17:25 +0000
Labels:             pod-template-hash=2660130850
                    app=firstpod
Annotations:        cni.projectcalico.org/podIP: 192.168.200.65/32
Status:             Running
IP:                 192.168.200.65
Controlled By:      ReplicaSet/firstpod-6bb4574d94

<output_omitted>
```

4. Note that the resources are in the default namespace. Get a list of available namespaces.

```
student@ckad-1:~$ kubectl get namespaces
```

```
NAME               STATUS     AGE
default            Active     20m
kube-node-lease    Active     20m
kube-public        Active     20m
kube-system        Active     20m
```

5. There are four default namespaces. Look at the pods in the `kube-system` namespace.

```
student@ckad-1:~$ kubectl get pod -n kube-system
```

```
NAME                                READY   STATUS    RESTARTS   AGE
calico-node-5ftrr                   2/2     Running   0          24m
calico-node-f7zrw                   2/2     Running   0          21m
coredns-fb8b8dccf-cmkds             1/1     Running   0          24m
coredns-fb8b8dccf-grltk             1/1     Running   0          24m
etcd-v141-r24p                      1/1     Running   0          23m
<output_omitted>
```

6. Now look at the pods in a namespace that does not exist. Note you do not receive an error.

```
student@ckad-1:~$ kubectl get pod -n fakenamespace
```

```
No resources found in fakenamespaces namespace.
```

7. You can also view resources in all namespaces at once. Use the `--all-namespaces` options to select objects in all namespaces at once.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
student@ckad-1:~$ kubectl get pod --all-namespaces
```

```
NAMESPACE     NAME                          READY   STATUS    RESTARTS   AGE
default       firstpod-69cfdfd8d9-kj6ql     1/1     Running   0          44m
kube-system   calico-node-5ftrr             2/2     Running   0          92m
kube-system   calico-node-f7zrw             2/2     Running   0          89m
kube-system   coredns-fb8b8dccf-cmkds       1/1     Running   0          92m
<output_omitted>
```

8. View several resources at once. Note that most resources have a short name such as `rs` for ReplicaSet, `po` for Pod, `svc` for Service, and `ep` for endpoint. Note the endpoint still exists after we deleted the pod.

```
student@ckad-1:~$ kubectl get deploy,rs,po,svc,ep
```

```
NAME                      READY  UP-TO-DATE  AVAILABLE  AGE
deployment.apps/firstpod  1/1    1           1          4m

NAME                                        DESIRED   CURRENT   READY....
replicaset.apps/firstpod-6bb4574d94-rqk76   1         1         1 ....

NAME                          READY  STATUS    RESTARTS   AGE
pod/firstpod-6bb4574d94-rqk76 1/1    Running   0          4m

NAME                 TYPE       CLUSTER-IP    EXTERNAL-IP PORT(S)      AGE
service/basicservice NodePort   10.108.147.76 <none>      80:31601/TCP 21m
service/kubernetes   ClusterIP  10.96.0.1     <none>      443/TCP      21m

NAME                   ENDPOINTS         AGE
endpoints/basicservice <none>            21m
endpoints/kubernetes   10.128.0.3:6443   21m
```

9. Delete the `ReplicaSet` and view the resources again. Note that the age on the `ReplicaSet` and the pod it controls is now less than a minute of age. The deployment operator started a new `ReplicaSet` operator when we deleted the existing one. The new `ReplicaSet` started another pod when the desired spec did not match the current status.

```
student@ckad-1:~$ kubectl delete rs firstpod-6bb4574d94-rqk76
```

```
replicaset.apps "firstpod-6bb4574d94-rqk76" deleted
```

```
student@ckad-1:~$ kubectl get deployment,rs,po,svc,ep
```

```
NAME                            READY  UP-TO-DATE AVAILABLE AGE
deployment.apps/firstpod  1/1    1          1         7m

NAME                                             DESIRED   CURRENT....
replicaset.apps/firstpod-6bb4574d94-rqk76   1         1     ....

NAME                           READY     STATUS    RESTARTS   AGE
pod/firstpod-7d99ffc75-p9hbw   1/1       Running   0          12s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   24m

NAME                   ENDPOINTS         AGE
endpoints/kubernetes   10.128.0.2:6443   80m
endpoints/basicservice   <none>            21m
```

10. This time delete the top-level controller. After about 30 seconds for everything to shut down you should only see the cluster service and endpoint remain for the cluster and the service we created.

```
student@ckad-1:~$ kubectl delete deployment firstpod
```

© Copyright the Linux Foundation 2020. All rights reserved.

```
deployment.apps "firstpod" deleted
```

```
student@ckad-1:~$ kubectl get deployment,rs,po,svc,ep
```

```
NAME                   TYPE      CLUSTER-IP     EXTERNAL-IP PORT(S)      AGE
service/basicservice NodePort  10.108.147.76 <none>      80:31601/TCP 35m
kubernetes             ClusterIP 10.96.0.1      <none>      443/TCP      24m

NAME                    ENDPOINTS          AGE
endpoints/basicservice <none>              21m
kubernetes              10.128.0.3:6443   24m
```

11. As we won't need it for a while, delete the `basicservice` service as well.

```
student@ckad-1:~$ kubectl delete svc basicservice
```

```
service "basicservice" deleted
```

Exercise 2.6: Domain Review

Very Important

The source pages and content in this review could change at any time. **IT IS YOUR RESPONSIBILITY TO CHECK THE CURRENT INFORMATION**.

1. Using a browser go to https://www.cncf.io/certification/ckad/ and read through the program description.

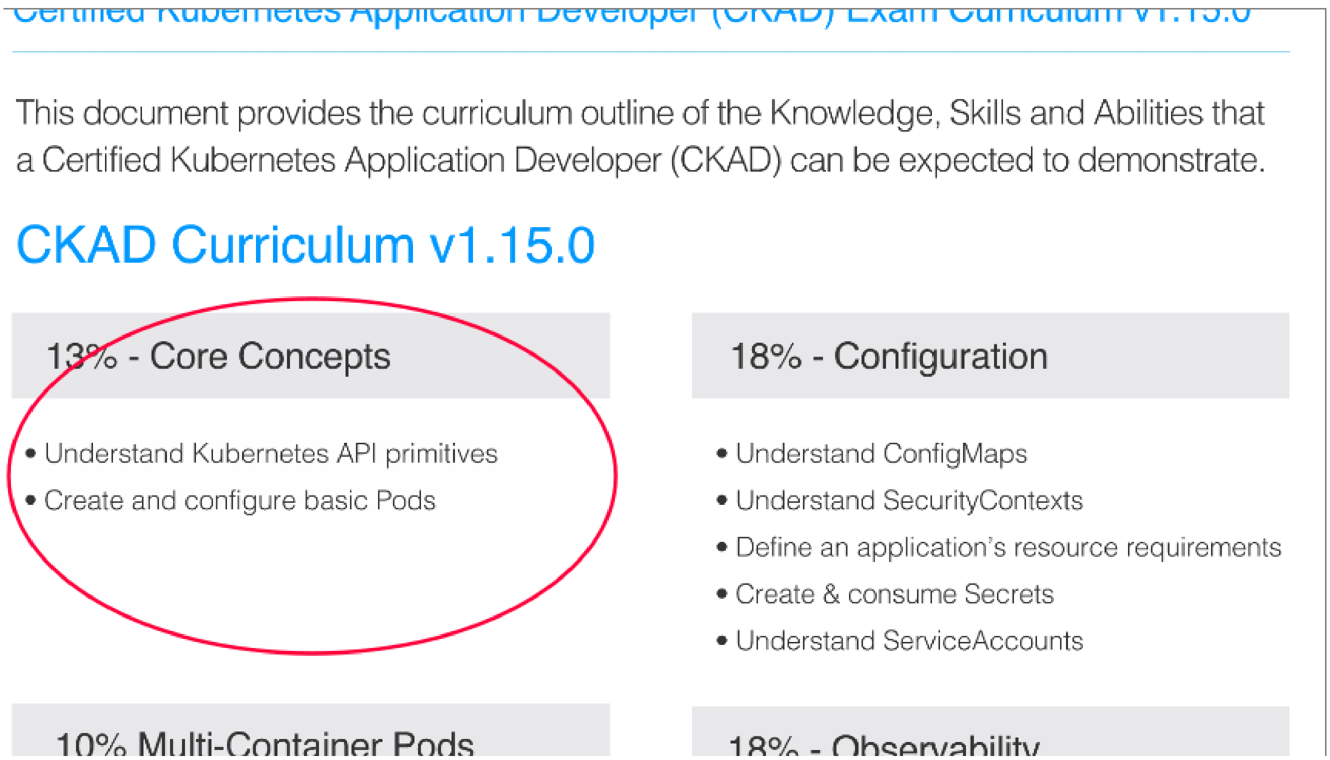
2. In the **Exam Resources** section open the Curriculum Overview and Candidate-handbook in new tabs. Both of these should be read and understood prior to sitting for the exam.

3. Navigate to the Curriculum Overview tab. You should see links for domain information for various versions of the exam. Select the latest version, such as **CKAD_Curriculum_V1.15.0.pdf**. The versions you see may be different. You should see a new page showing a PDF.

4. Read through the document. Be aware that the term `Understand`, such as `Understand Services`, is more than just knowing they exist. In this case expect it to also mean create, update, and troubleshoot.

5. Locate the **Core Concepts** section. If you review the lab, you will see we have covered these steps. Again, please note this document will change, distinct from this book. **It remains your responsibility to check for changes in the online document**. They may change on an irregular and unannounced basis.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE



Certified Kubernetes Application Developer (CKAD) Exam Curriculum v1.15.0

This document provides the curriculum outline of the Knowledge, Skills and Abilities that a Certified Kubernetes Application Developer (CKAD) can be expected to demonstrate.

CKAD Curriculum v1.15.0

13% - Core Concepts

- Understand Kubernetes API primitives
- Create and configure basic Pods

18% - Configuration

- Understand ConfigMaps
- Understand SecurityContexts
- Define an application's resource requirements
- Create & consume Secrets
- Understand ServiceAccounts

10% Multi-Container Pods

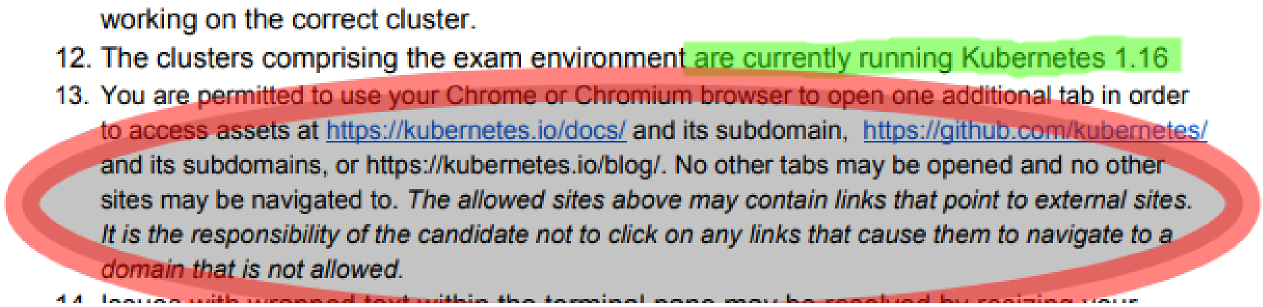
18% - Observability

Figure 2.1: **Core Concepts Domain**

6. Navigate to the Candidate-handbook tab. You are strongly encourage to read and understand this entire document prior to taking the exam. Again, please note this document will change, distinct from this book. **It remains your responsibility to check for changes in the online document**. They may change on an irregular and unannounced basis.

7. Find the **Guidelines and Tips for Use of the Linux server terminal**" section in the document.

8. Among other points you will note the current exam version and three (at the time this was written) domains and subdomains you can use, with some stated conditions.



working on the correct cluster.

12. The clusters comprising the exam environment are currently running Kubernetes 1.16
13. You are permitted to use your Chrome or Chromium browser to open one additional tab in order to access assets at https://kubernetes.io/docs/ and its subdomain, https://github.com/kubernetes/ and its subdomains, or https://kubernetes.io/blog/. No other tabs may be opened and no other sites may be navigated to. *The allowed sites above may contain links that point to external sites. It is the responsibility of the candidate not to click on any links that cause them to navigate to a domain that is not allowed.*
14. Issues with wrapped text within the terminal pane may be resolved by resizing your

Figure 2.2: **Exam Handbook Guidelines and Tips**

9. Using only the allowed browser, URLs, and subdomains search for and bookmark a YAML example to create and configure a basic pod. Ensure it works for the version of the exam you are taking. URLs may change, plan on checking each book mark prior to taking the exam.

10. Using a timer and bookmarked YAML files see how long it takes you to create and verify. Try it again and see how much faster you can complete and test each step:

    - A new pod with the **nginx** image. Showing all containers running and a `Ready` status.
    - A new service exposing the pod as a **nodePort**, which presents a working webserver configured in the previous step.

© Copyright the Linux Foundation 2020. All rights reserved.

- Update the pod to run the **nginx:1.11-alpine** image and re-verify you can view the webserver via a nodePort.

11. Find and use the `architecture-review1.yaml` file included in the course tarball. Your path, such as course number, may be different than the one in the example below. Use the **find** output. Determine if the pod is running. Fix any errors you may encounter. The use of **kubectl describe** may be helpful.

```
student@ckad-1:~$ find $HOME -name architecture-review1.yaml
```

```
/home/student/LFD259/SOLUTIONS/s_02/architecture-review1.yaml
```

```
student@ckad-1:~$ cp <copy-paste-from-above>  .

student@ckad-1:~$ kubectl create -f architecture-review1.yaml
```

12. Remove any pods or services you may have created as part of the review before moving on to the next section. For example:

```
student@ckad-1:~$ kubectl delete -f architecture-review1.yaml
```

Ghazela Technology
Academy

Copyright © 2020   adresse : 23, rue el Yakoute ,cité la Ghazelle ,Ariana
MF :1469595/X/A/M/000 – RC : B01912050
Contact : contact at ghazelatc.com
Téléphone :+216 71 866 142 , +216 27 862 155, +216 54 828 018

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

Ghazela Technology
Academy

Copyright © 2020 adresse : 23, rue el Yakoute ,cité la Ghazelle ,Ariana
MF :1469595/X/A/M/000 – RC : B01912050
Contact : contact at ghazelatc.com
Téléphone :+216 71 866 142 , +216 27 862 155, +216 54 828 018

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

Chapter 3

Build

Exercise 3.1: Deploy a New Application

Overview

In this lab we will deploy a very simple **Python** application, test it using Docker, ingest it into Kubernetes and configure probes to ensure it continues to run. This lab requires the completion of the previous lab, the installation and configuration of a Kubernetes cluster.

Working with Python

1. Install python on your master node. It may already be installed, as is shown in the output below.

```
student@ckad-1:~$ sudo apt-get -y install python
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
python is already the newest version (2.7.12-1~16.04).
python set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 5 not upgraded.
```

2. Locate the python binary on your system.

```
student@ckad-1:~$ which python
```

```
/usr/bin/python
```

3. Create and change into a new directory. The Docker build process pulls everything from the current directory into the image file by default. Make sure the chosen directory is empty.

```
student@ckad-1:~$ mkdir app1
student@ckad-1:~$ cd app1
student@ckad-1:~/app1$ ls -l
```

```
total 0
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

4. Create a simple python script which prints the time and hostname every 5 seconds. There are six commented parts to this script, which should explain what each part is meant to do. The script is included with others in the course tar file, though you are encouraged to create the file by hand if not already familiar with the process. While the command shows **vim** as an example other text editors such as **nano** work just as well.

```
student@ckad-1:˜/app1$ vim simple.py
```

simple.py

```
#!/usr/bin/python
## Import the necessary modules
import time
import socket

## Use an ongoing while loop to generate output
while True :

## Set the hostname and the current date
  host = socket.gethostname()
  date = time.strftime("%Y-%m-%d %H:%M:%S")

## Convert the date output to a string
  now = str(date)

## Open the file named date in append mode
## Append the output of hostname and time
  f = open("date.out", "a" )
  f.write(now + "\n")
  f.write(host + "\n")
  f.close()

## Sleep for five seconds then continue the loop
  time.sleep(5)
```

5. Make the file executable and test that it works. Use `Ctrl-C` to interrupt the `while` loop after 20 or 30 seconds. The output will be sent to a newly created file in your current directory called `date.out`.

```
student@ckad-1:˜/app1$ chmod +x simple.py
student@ckad-1:˜/app1$ ./simple.py
```

```
^CTraceback (most recent call last):
  File "./simple.py", line 42, in <module>
    time.sleep(5)
KeyboardInterrupt
```

6. and timedate stamps.

```
student@ckad-1:˜/app1$ cat date.out
```

```
2018-03-22 15:51:38
ckad-1
2018-03-22 15:51:43
ckad-1
2018-03-22 15:51:48
ckad-1
<output_omitted>
```

7. Create a text file named `Dockerfile`.

© Copyright the Linux Foundation 2020. All rights reserved.

Very Important

The name is important: it cannot have a suffix.

We will use three statements, `FROM` to declare which version of Python to use, `ADD` to include our script and `CMD` to indicate the action of the container. Should you be including more complex tasks you may need to install extra libraries, shown commented out as `RUN pip install` in the following example.

```
student@ckad-1:~/app1$ vim Dockerfile
```

Dockerfile

```
FROM python:2
ADD simple.py /
## RUN pip install pystrich
CMD [ "python", "./simple.py" ]
```

8. Build the container. The output below shows mid-build as necessary software is downloaded. You will need to use **sudo** in order to run this command. After the three step process completes the last line of output should indicate success. Note the dot (.) at the end of the command indicates the current directory.

```
student@ckad-1:~/app1$ sudo docker build -t simpleapp .
```

```
Sending build context to Docker daemon 3.072 kB
Step 1/3 : FROM python:2
2: Pulling from library/python
4176fe04cefe: Pull complete
851356ecf618: Pull complete
6115379c7b49: Pull complete
aaf7d781d601: Extracting [================>         ] 54.03 MB/135 MB
40cf661a3cc4: Download complete
c582f0b73e63: Download complete
6c1ea8f72a0d: Download complete
7051a41ae6b7: Download complete
<output_omitted>
Successfully built c4e0679b9c36
```

9. Verify you can see the new image among others downloaded during the build process, installed to support the cluster, or you may have already worked with. The newly created `simpleapp` image should be listed first.

```
student@ckad-1:~/app1$ sudo docker images
```

```
REPOSITORY                 TAG      IMAGE ID       CREATED         SIZE
simpleapp                  latest   ba54e4910397   6 seconds ago   902MB
k8s.gcr.io/kube-proxy      v1.19.0  4e68534e24f6   7 days ago      117MB
k8s.gcr.io/kube-apiserver  v1.19.0  a595af0107f9   7 days ago      173MB
<output_omitted>
```

10. Use **sudo docker** to run a container using the new image. While the script is running you won't see any output and the shell will be occupied running the image in the background. After 30 seconds use **ctrl-c** to interrupt. The local `date.out` file will not be updated with new times, instead that output will be a file of the container image.

```
student@ckad-1:~$ sudo docker run simpleapp
```

```
^CTraceback (most recent call last):
  File "./simple.py", line 24, in <module>
    time.sleep(5)
KeyboardInterrupt
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

11. Locate the newly created `date.out` file. The following command should show two files of this name, the one created when we ran simple.py and another under `/var/lib/docker` when run via a Docker container.

```
student@ckad-1:~/app1$ sudo find / -name date.out
```

```
/home/student/app1/date.out
/var/lib/docker/overlay2/ee814320c900bd24fad0c5db4a258d3c2b78a19cde
629d7de7d27270d6a0c1f5/diff/date.out
```

12. View the contents of the `date.out` file created via Docker. Note the need for **sudo** as Docker created the file this time, and the owner is `root`. The long name is shown on several lines in the example, but would be a single line when typed or copied.

```
student@ckad-1:~/app1$ sudo tail \
  /var/lib/docker/overlay2/ee814320c900bd24fad0c5db4a258d3c2b78a19cde
629d7de7d27270d6a0c1f5/diff/date.out
```

```
2018-03-22 16:13:46
53e1093e5d39
2018-03-22 16:13:51
53e1093e5d39
2018-03-22 16:13:56
53e1093e5d39
```

Exercise 3.2: Configure A Local Docker Repo

While we could create an account and upload our application to hub.docker.com, thus sharing it with the world, we will instead create a local repository and make it available to the nodes of our cluster.

1. We'll need to complete a few steps with special permissions, for ease of use we'll become root using **sudo**.

```
student@ckad-1:~/app1$ cd
student@ckad-1:~$ sudo -i
```

2. Install the **docker-compose** software and utilities to work with the **nginx** server which will be deployed with the registry.

```
root@ckad-1:~# apt-get install -y docker-compose apache2-utils
```

```
<output_omitted>
```

3. Create a new directory for configuration information. We'll be placing the repository in the root filesystem. A better location may be chosen in a production environment.

```
root@ckad-1:~# mkdir -p /localdocker/data
root@ckad-1:~# cd /localdocker/
```

4. Create a Docker compose file. Inside is an entry for the **nginx** web server to handle outside traffic and a registry entry listening to loopback port 5000 for running a local Docker registry.

```
root@ckad-1:/localdocker# vim docker-compose.yaml
```

YAML docker-compose.yaml

```yaml
nginx:
  image: "nginx:1.17"
```

Ghazela Technology Academy

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```yaml
  ports:
    - 443:443
  links:
    - registry:registry
  volumes:
    - /localdocker/nginx/:/etc/nginx/conf.d
registry:
  image: registry:2
  ports:
    - 127.0.0.1:5000:5000
  environment:
    REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY: /data
  volumes:
    - /localdocker/data:/data
```

5. Use the **docker-compose up** command to create the containers declared in the previous step YAML file. This will capture the terminal and run until you use **ctrl-c** to interrupt. There should be five `registry_1` entries with info messages about memory and which port is being listened to. Once we're sure the Docker file works we'll convert to a Kubernetes tool. **Let it run. You will use ctrl-c in a few steps.**

```
root@ckad-1:/localdocker# docker-compose up
```

```
Pulling nginx (nginx:1.17)...
1.17: Pulling from library/nginx
2a72cbf407d6: Pull complete
f37cbdc183b2: Pull complete
78b5ad0b466c: Pull complete
Digest: sha256:edad623fc7210111e8803b4359ba4854e101bcca1fe7f46bd1d35781f4034f0c
Status: Downloaded newer image for nginx:1.17
Creating localdocker_registry_1
Creating localdocker_nginx_1
Attaching to localdocker_registry_1, localdocker_nginx_1
registry_1  | time="2018-03-22T18:32:37Z" level=warning msg="No HTTP secret provided - generated ran
<output_omitted>
```

6. Test that you can access the repository. Open a second terminal to the master node. Use the **curl** command to test the repository. It should return `{}`, but does not have a carriage-return so will be on the same line as the following prompt. You should also see the `GET` request in the first, captured terminal, without error. Don't forget the trailing slash. You'll see a "Moved Permanently" message if the path does not match exactly.

```
student@ckad-1:~/localdocker$ curl http://127.0.0.1:5000/v2/
```

```
{}student@ckad-1:~/localdocker$
```

7. Now that we know **docker-compose** format is working, ingest the file into Kubernetes using **kompose**. Use **ctrl-c** to stop the previous **docker-compose** command.

```
^CGracefully stopping... (press Ctrl+C again to force)
Stopping localdocker_nginx_1 ... done
Stopping localdocker_registry_1 ... done
```

8. Download the kompose binary and make it executable. The command can run on a single line. Note that the option following the dash is the letter as in **o**utput. Also that is a zero, not capital O (ohh) in the short URL. The short URL goes here: https://github.com/kubernetes/kompose/releases/download/v1.1.0/kompose-linux-amd64

```
root@ckad-1:/localdocker# curl -L https://bit.ly/2tN0bEa -o kompose
```

Ghazela Technology Academy

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   609    0   609    0     0   1963      0 --:--:-- --:--:-- --:--:--  1970
100 45.3M  100 45.3M    0     0  16.3M      0  0:00:02  0:00:02 --:--:-- 25.9M
```

```
root@ckad-1:/localdocker# chmod +x kompose
```

9. Move the binary to a directory in our $PATH. Then return to your non-root user.

```
root@ckad-1:/localdocker# mv ./kompose /usr/local/bin/kompose
root@ckad-1:/localdocker# exit
```

10. Create two physical volumes in order to deploy a local registry for Kubernetes. 200Mi for each should be enough for each of the volumes. Use the **hostPath** `storageclass` for the volumes.

    More details on how persistent volumes and persistent volume claims are covered in an upcoming chapter, `Deployment Configuration`.

```
student@ckad-1:~$ vim vol1.yaml
```

vol1.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  labels:
    type: local
  name: task-pv-volume
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 200Mi
  hostPath:
    path: /tmp/data
  persistentVolumeReclaimPolicy: Retain
```

```
student@ckad-1:~$ vim vol2.yaml
```

vol2.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  labels:
    type: local
  name: registryvm
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 200Mi
  hostPath:
    path: /tmp/nginx
  persistentVolumeReclaimPolicy: Retain
```

Ghazela Technology Academy

11. Create both volumes.

© Copyright the Linux Foundation 2020. All rights reserved.

```
student@ckad-1:~$ kubectl create -f vol1.yaml
```

```
persistentvolume/task-pv-volume created
```

```
student@ckad-1:~$ kubectl create -f vol2.yaml
```

```
persistentvolume/registryvm created
```

12. Verify both volumes have been created. They should show an `Available` status.

```
student@ckad-1:~$ kubectl get pv
```

```
NAME             CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
  CLAIM      STORAGECLASS   REASON     AGE
registryvm       200Mi      RWO            Retain           Available
                                       27s
task-pv-volume   200Mi      RWO            Retain           Available
                                       32s
```

13. Go to the configuration file directory for the local Docker registry.

```
student@ckad-1:~$ cd /localdocker/
student@ckad-1:~/localdocker$ ls
```

```
data  docker-compose.yaml  nginx
```

14. Convert the Docker file into a single YAML file for use with Kubernetes. Not all objects convert exactly from Docker to **kompose**, you may get errors about the mount syntax for the new volumes. They can be safely ignored.

```
student@ckad-1:~/localdocker$ sudo kompose convert -f docker-compose.yaml -o localregistry.yaml
```

```
WARN Volume mount on the host "/localdocker/nginx/" isn't supported - ignoring path on the host
WARN Volume mount on the host "/localdocker/data" isn't supported - ignoring path on the host
```

15. Review the file. You'll find that multiple Kubernetes objects will have been created such as `Services`, `Persistent Volume Claims` and `Deployments` using environmental parameters and volumes to configure the container within.

```
student@ckad-1:/localdocker$ less localregistry.yaml
```

```
apiVersion: v1
items:
- apiVersion: v1
  kind: Service
  metadata:
    annotations:
      kompose.cmd: kompose convert -f docker-compose.yaml -o localregistry.yaml
      kompose.version: 1.1.0 (36652f6)
    creationTimestamp: null
    labels:
<output_omitted>
```

Ghazela Technology
Academy

16. View the cluster resources prior to deploying the registry. Only the cluster service and two available persistent volumes should exist in the default namespace.

```
student@ckad-1:~/localdocker$ kubectl get pods,svc,pvc,pv,deploy
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

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

NAME                            CAPACITY   ACCESS MODES   RECLAIM POLICY
STATUS      CLAIM     STORAGECLASS   REASON    AGE
persistentvolume/registryvm      200Mi      RWO            Retain
 Available                                 15s
persistentvolume/task-pv-volume 200Mi      RWO            Retain
 Available                                 17s
```

17. To illustrate the fast changing nature of Kubernetes you will show that the API has changed for `Deployments`. With each new release of Kubernetes you may want to plan on a YAML review. First determine new object settings and configuration, then compare and contrast to your existing YAML files. Edit and test for the new configurations.

    Another more common way to find YAML issues is to attempt to create an object using previous YAML in a new version of Kubernetes and track down errors. Not suggested, but what often happens instead of the following process.

    To view the current cluster requirements use the `--dry-run` option for the **kubectl create** command to see what the API now uses. We can compare the current values to our existing (previous version) YAML files. This will help determine what to edit for the local registry in an upcoming step.

```
student@ckad-1:~/localdocker$ kubectl create deployment drytry --image=nginx --dry-run=client -o yaml
```

drytry

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: drytry
  name: drytry
spec:
  replicas: 1
  selector:
    matchLabels:
      app: drytry
  strategy: {}
  template:
<output_omitted>
```

18. From the previous command output and comparing line by line to objects in the existing `localregistry.yaml` file output we can see that the `apiVersion` of the `Deployment` object has changed, and we need to add `selector`, add `matchLabels`, and a `label` line. The three lines to add will be part of the replicaSet information, right after the `replicas` line, with `selector` the same indentation as `replicas`.

    Following is a **diff** output, a common way to compare two files to each other, before and after an edit. Use the **man** page to decode the output if you are not already familiar with the command.

Ghazela Technology Academy

```
student@ckad-1:~/localdocker$ sudo cp localregistry.yaml old-localregistry.yaml

student@ckad-1:~/localdocker$ sudo vim localregistry.yaml
<make edits>

student@ckad-1:~/localdocker$ diff localregistry.yaml old-localregistry.yaml
```

```
41c41
< - apiVersion: apps/v1
---
> - apiVersion: extensions/v1beta1
```

© Copyright the Linux Foundation 2020. All rights reserved.

```
53,55d52
<     selector:
<       matchLabels:
<         io.kompose.service: nginx
93c90
< - apiVersion: apps/v1
---
> - apiVersion: extensions/v1beta1
105,107d101
<     selector:
<       matchLabels:
<         io.kompose.service: registry
```

19. Use **kubectl** to create the local docker registry.

```
student@ckad-1:~/localdocker$ kubectl create -f localregistry.yaml
```

```
service/nginx created
service/registry created
deployment.apps/nginx created
persistentvolumeclaim/nginx-claim0 created
deployment.apps/registry created
persistentvolumeclaim/registry-claim0 created
```

20. View the newly deployed resources. The persistent volumes should now show as `Bound`. Be aware that due to the manner that volumes are bound it is possible that the registry claim may not to be bound to the registry volume. Find the `service IP` for the registry. It should be sharing port 5000. In the example below the IP address is 10.110.186.162, yours may be different.

```
student@ckad-1:~/localdocker$ kubectl get pods,svc,pvc,pv,deploy
```

```
NAME                            READY     STATUS    RESTARTS   AGE
pod/nginx-6b58d9cdfd-95zxq      1/1       Running   0          1m
pod/registry-795c6c8b8f-b8z4k   1/1       Running   0          1m

NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
service/kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP    1h
service/nginx        ClusterIP   10.106.82.218   <none>        443/TCP    1m
service/registry     ClusterIP   10.110.186.162   <none>       5000/TCP   1m

NAME                                    STATUS    VOLUME
 CAPACITY   ACCESS MODES   STORAGECLASS   AGE
persistentvolumeclaim/nginx-claim0      Bound     registryvm
 200Mi      RWO                           1m
persistentvolumeclaim/registry-claim0   Bound     task-pv-volume
 200Mi      RWO                           1m

NAME                            CAPACITY   ACCESS MODES   RECLAIM POLICY
   STATUS    CLAIM    STORAGECLASS   REASON    AGE
persistentvolume/registryvm     200Mi      RWO            Retain
   Bound
default/nginx-claim0                                   5m
persistentvolume/task-pv-volume 200Mi      RWO            Retain
  Bound
default/registry-claim0                                6m

NAME                             READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx      1/1     1            1          12s
deployment.apps/registry   1/1     1            1          12s
```

Ghazela Technology Academy

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

21. Verify you get the same {} response using the Kubernetes deployed registry as we did when using **docker-compose**. Note you must use the trailing slash after v2. Please also note that if the connection hangs it may be due to a firewall issue. If running your nodes using GCE ensure your instances are using VPC setup and all ports are allowed. If using AWS also make sure all ports are being allowed.

    Edit the IP address to that of your registry service.

```
student@ckad-1:~/localdocker$ curl http://10.110.186.162:5000/v2/
```

```
{}student@ckad-1:~/localdocker$
```

22. Edit the Docker configuration file to allow insecure access to the registry. In a production environment steps should be taken to create and use TLS authentication instead. Use the IP and port of the registry you verified in the previous step.

```
student@ckad-1:~$ sudo vim /etc/docker/daemon.json
```

```
{ "insecure-registries":["10.110.186.162:5000"] }
```

23. Restart docker on the local system. It can take up to a minute for the restart to take place. Ensure the service is active. It should report that the service recently became status as well.

```
student@ckad-1:~$ sudo systemctl restart docker.service
student@ckad-1:~$ sudo systemctl status docker.service | grep Active
```

```
   Active: active (running) since Tue 2019-09-24 15:24:36 UTC; 40s ago
```

24. Download and tag a typical image from hub.docker.com. Tag the image using the IP and port of the registry. We will also use the latest tag.

```
student@ckad-1:~$ sudo docker pull ubuntu
```

```
Using default tag: latest
latest: Pulling from library/ubuntu
<output_omitted>
Digest: sha256:9ee3b83bcaa383e5e3b657f042f4034c92cdd50c03f73166c145c9ceaea9ba7c
Status: Downloaded newer image for ubuntu:latest
```

```
student@ckad-1:~$ sudo docker tag ubuntu:latest 10.110.186.162:5000/tagtest
```

25. Push the newly tagged image to your local registry. If you receive an error about an HTTP request to an HTTPS client check that you edited the /etc/docker/daemon.json file correctly and restarted the service.

```
student@ckad-1:~$ sudo docker push 10.110.186.162:5000/tagtest
```

```
The push refers to a repository [10.110.186.162:5000/tagtest]
db584c622b50: Pushed
52a7ea2bb533: Pushed
52f389ea437e: Pushed
88888b9b1b5b: Pushed
a94e0d5a7c40: Pushed
latest: digest: sha256:0847cc7fed1bfafac713b0aa4ddfb8b9199a99092ae1fc4e718cb28e8528f65f size: 1357
```

Ghazela Technology Academy

26. We will test to make sure we can also pull images from our local repository. Begin by removing the local cached images.

```
student@ckad-1:~$ sudo docker image remove ubuntu:latest
```

```
Untagged: ubuntu:latest
Untagged: ubuntu@sha256:e348fbbea0e0a0e73ab0370de151e7800684445c509d46195aef73e090a49bd6
```

```
student@ckad-1:~$ sudo docker image remove 10.110.186.162:5000/tagtest
```

© Copyright the Linux Foundation 2020. All rights reserved.

```
Untagged: 10.110.186.162:5000/tagtest:latest
<output_omitted>
```

27. Pull the image from the local registry. It should report the download of a newer image.

```
student@ckad-1:~$ sudo docker pull 10.110.186.162:5000/tagtest
```

```
Using default tag: latest
latest: Pulling from tagtest
Digest: sha256:0847cc7fed1bfafac713b0aa4ddfb8b9199a99092ae1fc4e718cb28e8528f65f
Status: Downloaded newer image for 10.110.186.162:5000/tagtest:latest
```

28. Use docker tag to assign the `simpleapp` image and then push it to the local registry. The image and dependent images should be pushed to the local repository.

```
student@ckad-1:~$ sudo docker tag simpleapp 10.110.186.162:5000/simpleapp
student@ckad-1:~$ sudo docker push 10.110.186.162:5000/simpleapp
```

```
The push refers to a repository [10.110.186.162:5000/simpleapp]
321938b97e7e: Pushed
ca82a2274c57: Pushed
de2fbb43bd2a: Pushed
4e32c2de91a6: Pushed
6e1b48dc2ccc: Pushed
ff57bdb79ac8: Pushed
6e5e20cbf4a7: Pushed
86985c679800: Pushed
8fad67424c4e: Pushed
latest: digest: sha256:67ea3e11570042e70cdcbad684a1e2986f59aaf53703e51725accdf5c70d475a size: 2218
```

29. Configure the worker (second) node to use the local registry running on the master server. Connect to the worker node. Edit the Docker `daemon.json` file with the same values as the master node and restart the service. Ensure it is `active`.

```
student@ckad-2:~$ sudo vim /etc/docker/daemon.json
```

```
{ "insecure-registries":["10.110.186.162:5000"] }
```

```
student@ckad-2:~$ sudo systemctl restart docker.service

student@ckad-2:~$ sudo systemctl status docker.service
```

30. From the worker node, pull the recently pushed image from the registry running on the master node.

```
student@ckad-2:~$ sudo docker pull 10.110.186.162:5000/simpleapp
```

```
Using default tag: latest
latest: Pulling from simpleapp
f65523718fc5: Pull complete
1d2dd88bf649: Pull complete
c09558828658: Pull complete
0e1d7c9e6c06: Pull complete
c6b6fe164861: Pull complete
45097146116f: Pull complete
f21f8abae4c4: Pull complete
1c39556edcd0: Pull complete
85c79f0780fa: Pull complete
Digest: sha256:67ea3e11570042e70cdcbad684a1e2986f59aaf53703e51725accdf5c70d475a
Status: Downloaded newer image for 10.110.186.162:5000/simpleapp:latest
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

31. Return to the master node and deploy the `simpleapp` in Kubernetes with several replicas. We will name the deployment `try1`. Scale to have six replicas. Multiple replicas the scheduler should run some containers on each node.

```
student@ckad-1:~$ kubectl create deployment try1 --image=10.110.186.162:5000/simpleapp
```

```
deployment.apps/try1 created
```

```
student@ckad-1:~$  kubectl scale deployment try1 --replicas=6
```

```
deployment.apps/try1 scaled
```

32. View the running pods. You should see six replicas of `simpleapp` as well as two running the locally hosted image repository.

```
student@ckad-1:~$ kubectl get pods
```

```
NAME                        READY     STATUS    RESTARTS    AGE
nginx-6b58d9cdfd-j6jm6      1/1       Running   1           13m
registry-795c6c8b8f-5jnpn   1/1       Running   1           13m
try1-857bdcd888-6klrr       1/1       Running   0           25s
try1-857bdcd888-9pwnp       1/1       Running   0           25s
try1-857bdcd888-9xkth       1/1       Running   0           25s
try1-857bdcd888-tw58z       1/1       Running   0           25s
try1-857bdcd888-xj9lk       1/1       Running   0           25s
try1-857bdcd888-znpm8       1/1       Running   0           25s
```

33. On the `second node` use **sudo docker ps** to verify containers of `simpleapp` are running. The scheduler will usually balance pod count across nodes. As the master already has several pods running the new pods may be on the worker.

```
student@ckad-2:~$ sudo docker ps | grep simple
```

```
3ae4668d71d8     \
  10.110.186.162:5000/simpleapp@sha256:67ea3e11570042e70cdcbad684a1e2986f59aaf53703e51725accdf5c70d475a \
            "python ./simple.py"     48 seconds ago      Up 48 seconds \
             k8s_try1_try1-857bdcd888-9xkth_default_2e94b97e-322a-11e8-af56-42010a800004_0
ef6448764625 \
  10.110.186.162:5000/simpleapp@sha256:67ea3e11570042e70cdcbad684a1e2986f59aaf53703e51725accdf5c70d475a \
            "python ./simple.py"     48 seconds ago      Up 48 seconds \
             k8s_try1_try1-857bdcd888-znpm8_default_2e99f356-322a-11e8-af56-42010a800004_0
```

34. Return to the `master node`. Save the `try1` deployment as YAML.

```
student@ckad-1:~/app1$ cd ~/app1/
student@ckad-1:~/app1$ kubectl get deployment try1 -o yaml > simpleapp.yaml
```

Ghazela Technology Academy

35. Delete and recreate the `try1` deployment using the YAML file. Verify the deployment is running with the expected six replicas.

```
student@ckad-1:~$ kubectl delete deployment try1
```

```
deployment.apps "try1" deleted
```

```
student@ckad-1:~/app1$ kubectl create -f simpleapp.yaml
```

```
deployment.apps/try1 created
```

```
student@ckad-1:~/app1$ kubectl get deployment
```

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
NAME       READY   UP-TO-DATE   AVAILABLE   AGE
nginx      1/1     1            1           15m
registry   1/1     1            1           15m
try1       6/6     6            6           5s
```

Exercise 3.3: Configure Probes

> When large datasets need to be loaded or a complex application launched prior to client access, a `readinessProbe` can be used. The pod will not become available to the cluster until a test is met and returns a successful exit code. Both `readinessProbes` and `livenessProbes` use the same syntax and are identical other than the name. Where the `readinessProbe` is checked prior to being ready, then not again, the `livenessProbe` continues to be checked.
>
> There are three types of liveness probes: a command returns a zero exit value, meaning success, an HTTP request returns a response code in the 200 to 399 range, and the third probe uses a TCP socket. In this example we'll use a command, **cat**, which will return a zero exit code when the file `/tmp/healthy` has been created and can be accessed.

1. Edit the YAML deployment file and add the stanza for a `readinessprobe`. Remember that when working with YAML whitespace matters. Indentation is used to parse where information should be associated within the stanza and the entire file. Do not use tabs. If you get an error about validating data, check the indentation. It can also be helpful to paste the file to this website to see how indentation affects the JSON value, which is actually what Kubernetes ingests: https://www.json2yaml.com/ An edited file is also included in the tarball, but requires the image name to be edited to match your registry IP address.

   ```
   student@ckad-1:~/app1$ vim simpleapp.yaml
   ```

   simpleapp.yaml

   ```
   ....
       spec:
         containers:
         - image: 10.111.235.60:5000/simpleapp:latest
           imagePullPolicy: Always
           name: simpleapp
           readinessProbe:         #<--This line and next five
             periodSeconds: 5
             exec:
               command:
               - cat
               - /tmp/healthy
           resources: {}
   ....
   ```

Ghazela Technology Academy

2. Delete and recreate the `try1` deployment.

   ```
   student@ckad-1:~/app1$ kubectl delete deployment try1
   ```

   ```
   deployment.apps "try1" deleted
   ```

   ```
   student@ckad-1:~/app1$ kubectl create -f simpleapp.yaml
   ```

   ```
   deployment.apps/try1 created
   ```

3. The new `try1` deployment should reference six pods, but show zero available. They are all missing the `/tmp/healthy` file.

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
student@ckad-1:~/app1$ kubectl get deployment
```

```
NAME       READY   UP-TO-DATE   AVAILABLE   AGE
nginx      1/1     1            1           19m
registry   1/1     1            1           19m
try1       0/6     6            0           15s
```

4. Take a closer look at the pods. Choose one of the `try1` pods as a test to create the health check file.

```
student@ckad-1:~/app1$ kubectl get pods
```

```
NAME                          READY     STATUS    RESTARTS    AGE
nginx-6b58d9cdfd-g7lnk        1/1       Running   1           40m
registry-795c6c8b8f-7vwdn     1/1       Running   1           40m
try1-9869bdb88-2wfnr          0/1       Running   0           26s
try1-9869bdb88-6bknl          0/1       Running   0           26s
try1-9869bdb88-786v8          0/1       Running   0           26s
try1-9869bdb88-gmvs4          0/1       Running   0           26s
try1-9869bdb88-lfvlx          0/1       Running   0           26s
try1-9869bdb88-rtchc          0/1       Running   0           26s
```

5. Run the bash shell interactively and touch the `/tmp/healthy` file.

```
student@ckad-1:~/app1$ kubectl exec  -it try1-9869bdb88-rtchc -- /bin/bash

root@try1-9869bdb88-rtchc:/# touch /tmp/healthy

root@try1-9869bdb88-rtchc:/# exit
```

```
exit
```

6. Wait at least five seconds, then check the pods again. Once the probe runs again the container should show available quickly. The pod with the existing `/tmp/healthy` file should be running and show `1/1` in a `READY` state. The rest will continue to show `0/1`.

```
student@ckad-1:~/app1$ kubectl get pods
```

```
NAME                          READY     STATUS    RESTARTS    AGE
nginx-6b58d9cdfd-g7lnk        1/1       Running   1           44m
registry-795c6c8b8f-7vwdn     1/1       Running   1           44m
try1-9869bdb88-2wfnr          0/1       Running   0           4m
try1-9869bdb88-6bknl          0/1       Running   0           4m
try1-9869bdb88-786v8          0/1       Running   0           4m
try1-9869bdb88-gmvs4          0/1       Running   0           4m
try1-9869bdb88-lfvlx          0/1       Running   0           4m
try1-9869bdb88-rtchc          1/1       Running   0           4m
```

Ghazela Technology Academy

7. Touch the file in the remaining pods. Consider using a **for** loop, as an easy method to update each pod. Note the >shown in the output represents the secondary prompt, you would not type in that character

```
student@ckad-1:~$ for name in try1-9869bdb88-2wfnr try1-9869bdb88-6bknl \
> try1-9869bdb88-786v8 try1-9869bdb88-gmvs4 try1-9869bdb88-lfvlx
> do
> kubectl exec $name -- touch /tmp/healthy
> done
```

8. It may take a short while for the probes to check for the file and the health checks to succeed.

```
student@ckad-1:~/app1$ kubectl get pods
```

© Copyright the Linux Foundation 2020. All rights reserved.

```
NAME                        READY     STATUS    RESTARTS   AGE
nginx-6b58d9cdfd-g7lnk      1/1       Running   1          1h
registry-795c6c8b8f-7vwdn   1/1       Running   1          1h
try1-9869bdb88-2wfnr        1/1       Running   0          22m
try1-9869bdb88-6bknl        1/1       Running   0          22m
try1-9869bdb88-786v8        1/1       Running   0          22m
try1-9869bdb88-gmvs4        1/1       Running   0          22m
try1-9869bdb88-lfvlx        1/1       Running   0          22m
try1-9869bdb88-rtchc        1/1       Running   0          22m
```

9. Now that we know when a pod is healthy, we may want to keep track that it stays healthy, using a `livenessProbe`. You could use one probe to determine when a pod becomes available and a second probe, to a different location, to ensure ongoing health.

   Edit the deployment again. Add in a `livenessProbe` section as seen below. This time we will add a `Sidecar` container to the pod running a `simple` application which will respond to port 8080. Note that the dash (**-**) in front of the name. Also `goproxy` is indented the same number of spaces as the **-** in front of the `image:` line for `simpleapp` earlier in the file. In this example that would be seven spaces

   ```
   student@ckad-1:~/app1$ vim simpleapp.yaml
   ```

   simpleapp.yaml

   ```
    ....
            terminationMessagePath: /dev/termination-log
            terminationMessagePolicy: File
          - name: goproxy                 #<-- Indented 7 spaces, add lines from here...
            image: k8s.gcr.io/goproxy:0.1
            ports:
            - containerPort: 8080
            readinessProbe:
              tcpSocket:
                port: 8080
              initialDelaySeconds: 5
              periodSeconds: 10
            livenessProbe:                #<-- This line is 9 spaces indented, fyi
              tcpSocket:
                port: 8080
              initialDelaySeconds: 15
              periodSeconds: 20           #<-- ....to here
          dnsPolicy: ClusterFirst
          restartPolicy: Always
    ....
   ```

10. Delete and recreate the deployment.

    ```
    student@ckad-1:~$ kubectl delete deployment try1
    ```

    ```
    deployment.apps "try1" deleted
    ```

    ```
    student@ckad-1:~$ kubectl create -f simpleapp.yaml
    ```

    ```
    deployment.apps/try1 created
    ```

11. View the newly created pods. You'll note that there are two containers per pod, and only one is running. The new `simpleapp` containers will not have the `/tmp/healthy` file, so they will not become available until we touch the `/tmp/healthy` file again. We could include a command which creates the file into the container arguments. The output below shows it can take a bit for the old pods to terminate.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
student@ckad-1:~$ kubectl get pods
```

```
NAME                           READY     STATUS              RESTARTS    AGE
nginx-6b58d9cdfd-g7lnk         1/1       Running             1           13h
registry-795c6c8b8f-7vwdn      1/1       Running             1           13h
try1-76cc5ffcc6-4rjvh          1/2       Running             0           3s
try1-76cc5ffcc6-bk5f5          1/2       Running             0           3s
try1-76cc5ffcc6-d8n5q          0/2       ContainerCreating   0           3s
try1-76cc5ffcc6-mm6tw          1/2       Running             0           3s
try1-76cc5ffcc6-r9q5n          1/2       Running             0           3s
try1-76cc5ffcc6-tx4dz          1/2       Running             0           3s
try1-9869bdb88-2wfnr           1/1       Terminating         0           12h
try1-9869bdb88-6bknl           1/1       Terminating         0           12h
try1-9869bdb88-786v8           1/1       Terminating         0           12h
try1-9869bdb88-gmvs4           1/1       Terminating         0           12h
try1-9869bdb88-lfvlx           1/1       Terminating         0           12h
try1-9869bdb88-rtchc           1/1       Terminating         0           12h
```

12. Create the health check file for the `readinessProbe`. You can use a **for** loop again for each action, this setup will leverage labels so you don't have to look up the pod names. As there are now two containers in the pod, you should include the container name for which one will execute the command. If no name is given, it will default to the first container. Depending on how you edited the YAML file `try1` should be the first pod and `goproxy` the second. To ensure the correct container is updated, add **-c simpleapp** to the **kubectl** command. Your pod names will be different. Use the names of the newly started containers from the **kubectl get pods** command output. Note the >character represents the secondary prompt, you would not type in that character.

```
student@ckad-1:~$ for name in $(kubectl get pod -l app=try1 -o name)
> do
> kubectl exec $name -c simpleapp -- touch /tmp/healthy
> done
```

13. In the next minute or so the `Sidecar` container in each pod, which was not running, will change status to `Running`. Each should show 2/2 containers running.

```
student@ckad-1:~$ kubectl get pods
```

```
NAME                           READY     STATUS           RESTARTS    AGE
nginx-6b58d9cdfd-g7lnk         1/1       Running          1           13h
registry-795c6c8b8f-7vwdn      1/1       Running          1           13h
try1-76cc5ffcc6-4rjvh          2/2       Running          0           3s
try1-76cc5ffcc6-bk5f5          2/2       Running          0           3s
try1-76cc5ffcc6-d8n5q          2/2       Running          0           3s
try1-76cc5ffcc6-mm6tw          2/2       Running          0           3s
try1-76cc5ffcc6-r9q5n          2/2       Running          0           3s
try1-76cc5ffcc6-tx4dz          2/2       Running          0           3s
```

14. View the events for a particular pod. Even though both containers are currently running and the pod is in good shape, note the events section shows the issue.

```
student@ckad-1:~/app1$ kubectl describe pod try1-76cc5ffcc6-tx4dz | tail
```

```
  Normal   SuccessfulMountVolume  9m                kubelet, ckad-1-lab-x6dj
MountVolume.SetUp succeeded for volume "default-token-jf69w"
  Normal   Pulling                9m                kubelet, ckad-1-lab-x6dj
pulling image "10.108.143.90:5000/simpleapp"
  Normal   Pulled                 9m                kubelet, ckad-1-lab-x6dj
Successfully pulled image "10.108.143.90:5000/simpleapp"
  Normal   Created                9m                kubelet, ckad-1-lab-x6dj
Created container
  Normal   Started                9m                kubelet, ckad-1-lab-x6dj
Started container
  Normal   Pulling                9m                kubelet, ckad-1-lab-x6dj
```

© Copyright the Linux Foundation 2020. All rights reserved.

```
pulling image "k8s.gcr.io/goproxy:0.1"
  Normal   Pulled                 9m                kubelet, ckad-1-lab-x6dj
Successfully pulled image "k8s.gcr.io/goproxy:0.1"
  Normal   Created                9m                kubelet, ckad-1-lab-x6dj
Created container
  Normal   Started                9m                kubelet, ckad-1-lab-x6dj
Started container
  Warning  Unhealthy              4m (x60 over 9m)  kubelet, ckad-1-lab-x6dj
Readiness probe failed: cat: /tmp/healthy: No such file or directory
```

15. If you look for the status of each container in the pod, they should show that both are `Running` and ready showing `True`.

```
student@ckad-1:~/app1$ kubectl describe pod try1-76cc5ffcc6-tx4dz | grep -E 'State|Ready'
```

```
    State:          Running
    Ready:          True
    State:          Running
    Ready:          True
  Ready             True
  ContainersReady   True
```

Exercise 3.4: Domain Review

Very Important

The source pages and content in this review could change at any time. **IT IS YOUR RESPONSIBILITY TO CHECK THE CURRENT INFORMATION**.

Revisit the CKAD domain list on Curriculum Overview and locate some of the topics we have covered in this chapter.

- Understand Multi-Container Pod design patterns (e .g. ambassador, adapter, sidecar)
- Understand LivenessProbes and ReadinessProbes
- Understand container logging

Figure 3.1: **Observability Domain**

Focus on ensuing you have all the necessary files and processes understood first. Repeat the review until you are sure you have bookmarks of YAML samples and can complete each step quickly.

1. Using the three URL locations allowed by the exam, find and bookmark working YAML examples for LivenessProbes, ReadinessProbes, and multi-container pods.

2. Deploy a new nginx webserver. Add a LivenessProbe and a ReadinessProbe on port 80. Test that both probes and the webserver work.

3. Use the `build-review1.yaml` file to create a non-working deployment. Fix the deployment such that both containers are running and in a `READY` state. The web server listens on port 80, and the proxy listens on port 8080.

4. View the default page of the web server. When successful verify the `GET` activity logs in the container log. The message should look something like the following. Your time and IP may be different.

```
192.168.124.0 - - [30/Jan/2020:03:30:31 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.58.0" "-"
```

5. Remove any resources created in this review.

Ghazela Technology Academy

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

Ghazela Technology
Academy

Copyright © 2020   adresse : 23, rue el Yakoute ,cité la Ghazelle ,Ariana
MF :1469595/X/A/M/000 – RC : B01912050
Contact : contact at ghazelatc.com
Téléphone :+216 71 866 142 , +216 27 862 155, +216 54 828 018

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

Chapter 4

Design

Exercise 4.1: Planning the Deployment

Overview

In this exercise we will investigate common network plugins. Each **kubelet** agent uses one plugin at a time. Due to complexity, the entire cluster uses one plugin which is configured prior to application deployment. Some plugins don't honor security configurations such as network policies. Should you design a deployment which and use a network policy there wouldn't be an error; the policy would have no effect. While developers typically wouldn't care much about the mechanics of it can affect the availability of features and troubleshooting of newly decoupled microservices.

While still new, the community is moving towards the **Container Network Interface** (**CNI**) specification (https://github.com/containernetworking/cni). This provides the most flexibility and features in the fast changing space of container networking.

A common alternative is **kubenet**, a basic plugin which relies on the cloud provider to handle routing and cross-node networking. In a previous lab exercise we configured **Project Calico**. Classic and external modes are also possible. Several software defined network projects intended for Kubernetes have been created recently, with new features added regularly. Be aware that **Calico** is a dynamic project with ongoing and frequent changes.

Evaluate Network Plugins

1. Verify your nodes are using a CNI plugin. Read through the startup process of CNI. Each message begins with a time stamp, type of message and what is reporting the message.

```
student@ckad-1:~$ less /var/log/calico/cni/cni.log
```

```
2020-10-05 16:50:00.960 [INFO][21091] ipam.go 936: Attempting to assign 1 addresses
from block block=192.168.219.64/26 handle="k8s-pod-network.d303bf71efef750a3f420486f
e7b8b9e945abe841ef192673a44435af1c20428c" host="master"
```

2. There are many CNI providers possible. The following list represents some of the more common choices, but it is not exhaustive. With many new plugins being developed there may be another which better serves your needs. Use these websites to answer questions which follow. While we strive to keep the answers accurate, please be aware that this area has a lot of attention and development and changes often.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

- **Project Calico**
  https://docs.projectcalico.org/v3.0/introduction/
- **Calico with Canal**
  https://docs.projectcalico.org/v3.0/getting-started/kubernetes/installation/hosted/canal
- **Weave Works**
  https://www.weave.works/docs/net/latest/kubernetes/kube-addon
- **Flannel**
  https://github.com/coreos/flannel
- **Romana**
  http://romana.io/how/romana_basics/
- **Kube Router**
  https://www.kube-router.io
- **Kopeio**
  https://github.com/kopeio/networking

3. Which of the plugins allow vxlans?
4. Which are layer 2 plugins?
5. Which are layer 3?
6. Which allow network policies?
7. Which can encrypt all TCP and UDP traffic?

Multi-container Pod Considerations

Using the information learned from this chapter, consider the following questions:

1. Which deployment method would allow the most flexibility, multiple applications per pod or one per pod?
2. Which deployment method allows for the most granular scalability?
3. Which have the best performance?
4. How many IP addresses are assigned per pod?
5. What are some ways containers can communicate within the same pod?
6. What are some reasons you should have multiple containers per pod?

Do you really know?

When and why would you use a multi-container pod?

Have you found a YAML example online?

Go back and review multi-container pod types and content on decoupling if you can't easily answer these questions. We touched on adding a second logging and a readiness container in a previous chapter and will work more with logging a future exercise.

Solution 4.1

© Copyright the Linux Foundation 2020. All rights reserved.

Plugin Answers

1. Which of the plugins allow vxlans?

   Canal, Project Calico, Flannel, Kopeio-networking, Weave Net

2. Which are layer 2 plugins?

   Canal, Flannel, Kopeio-networking, Weave Net

3. Which are layer 3?

   Project Calico, Romana, Kube Router

4. Which allow network policies?

   Project Calico, Canal, Kube Router, Romana Weave Net

5. Which can encrypt all TCP and UDP traffic?

   Project Calico, Kopeio, Weave Net

Multi Pod Answers

1. Which deployment method would allow the most flexibility, multiple applications per pod or one per Pod?

   One per pod

2. Which deployment method allows for the most granular scalability?

   One per pod

3. Which have the best inter-container performance?

   Multiple per pod.

4. How many IP addresses are assigned per pod?

   One

5. What are some ways containers can communicate within the same pod?

   IPC, loopback or shared filesystem access.

6. What are some reasons you should have multiple containers per pod?

   Lean containers may not have functionality like logging. Able to maintain lean execution but add functionality as necessary, like Ambassadors and Sidecar containers.

Exercise 4.2: Designing Applications With Duration: Create a Job

> While most applications 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`

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

```
student@ckad-1:~$ vim job.yaml
```

job.yaml

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: sleepy
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```yaml
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@ckad-1:~$ kubectl create -f job.yaml
```

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

```
student@ckad-1:~$ kubectl get job
```

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

```
student@ckad-1:~$ 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:     Sun, 03 Nov 2019  04:22:50 +0000
Completed At:   Sun, 03 Nov 2019  04:22:55 +0000
Duration:       5s
Pods Statuses:  0 Running / 1 Succeeded / 0 Failed
<output_omitted>
```

```
student@ckad-1:~$ 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@ckad-1:~$ 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>
```

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

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

```
student@ckad-1:~$ kubectl delete jobs.batch sleepy
```

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

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

```
student@ckad-1:~$ 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@ckad-1:~$ kubectl create -f job.yaml
```

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

```
student@ckad-1:~$ 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@ckad-1:~$ kubectl get pods
```

```
NAME                         READY   STATUS           RESTARTS   AGE
nginx-67f8fb575f-g4468       1/1     Running          2          2d
registry-56cffc98d6-xlhhf    1/1     Running          1          2d
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@ckad-1:~$ kubectl get jobs
```

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

```
student@ckad-1:~$ 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@ckad-1:~$ vim job.yaml
```

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

job.yaml

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

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

```
student@ckad-1:~$ kubectl create -f job.yaml
```

```
student@ckad-1:~$ kubectl get pods
```

```
NAME                            READY   STATUS    RESTARTS   AGE
nginx-67f8fb575f-g4468          1/1     Running   2          2d
registry-56cffc98d6-xlhhf       1/1     Running   1          2d
sleepy-8xwpc                    1/1     Running   0          5s
sleepy-xjqnf                    1/1     Running   0          5s
try1-c9cb54f5d-b45gl            2/2     Running   0          8h
<output_omitted>
```

```
student@ckad-1:~$ 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.

```
student@ckad-1:~$ vim job.yaml
```

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: ["3"]
<output_omitted>
```

12. Delete and recreate the job again. It should run for four times then continue to age without further completions.

```
student@ckad-1:~$ kubectl delete jobs.batch sleepy
```

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

```
student@ckad-1:~$ kubectl create -f job.yaml
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

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

```
student@ckad-1:~$ kubectl get jobs
```

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

```
student@ckad-1:~$ kubectl get jobs
```

```
NAME     COMPLETIONS   DURATION   AGE
sleepy   4/5           16s        16s
```

13. View the `message:` entry in the `Status` section of the object YAML output. You may see less `status` if the job has yet to run. Wait and try again, if so.

```
student@ckad-1:~$ kubectl get job sleepy -o yaml
```

```
<output_omitted>
status:
  conditions:
  - lastProbeTime: "2019-11-03T16:06:10Z"
    lastTransitionTime: "2019-11-03T16:06:10Z"
    message: Job was active longer than specified deadline
    reason: DeadlineExceeded
    status: "True"
    type: Failed
  failed: 1
  startTime: "2019-11-03T16:05:55Z"
  succeeded: 4
```

14. Delete the job.

```
student@ckad-1:~$ kubectl delete jobs.batch sleepy
```

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

Exercise 4.3: Designing Applications With Duration: 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@ckad-1:~$ cp job.yaml cronjob.yaml
```

2. Edit the file to look like the annotated file shown below.

```
student@ckad-1:~$ vim cronjob.yaml
```

cronjob.yaml

```yaml
apiVersion: batch/v1beta1    #<-- Add beta1 to be v1beta1
kind: CronJob    #<-- Change this line
```

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```yaml
metadata:
  name: sleepy
spec:                        #<-- Remove completions:, parallelism:, and activeDeadlineSeconds:
  schedule: "*/2 * * * *"    #<-- Add Linux style cronjob syntax
  jobTemplate:               #<-- New jobTemplate and spec
    spec:
      template:   #<-- This and following lines space four to right
        spec:
          containers:
          - name: resting
            image: busybox
            command: ["/bin/sleep"]
            args: ["3"]
          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@ckad-1:˜$ kubectl create -f cronjob.yaml
```

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

```
student@ckad-1:˜$ kubectl get cronjobs.batch
```

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

```
student@ckad-1:˜$ kubectl get job
```

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

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

```
student@ckad-1:˜$ kubectl get cronjobs.batch
```

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

```
student@ckad-1:˜$ kubectl get jobs.batch
```

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

```
student@ckad-1:˜$ 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@ckad-1:˜$ vim cronjob.yaml
```

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

cronjob.yaml

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

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

```
student@ckad-1:~$ kubectl delete cronjobs.batch sleepy
```

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

```
student@ckad-1:~$ kubectl create -f cronjob.yaml
```

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

```
student@ckad-1:~$ sleep 120 ; kubectl get jobs
```

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

```
student@ckad-1:~$ kubectl get cronjobs.batch
```

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

```
student@ckad-1:~$ kubectl get jobs
```

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

```
student@ckad-1:~$ kubectl get jobs
```

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

```
student@ckad-1:~$ 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@ckad-1:~$  kubectl delete cronjobs.batch sleepy
```

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

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

Exercise 4.4: Using Labels

Create and work with labels. We will understand how the deployment, replicaSet, and pod labels interact.

1. Create a new deployment called `design2`

```
student@ckad-1:~$ kubectl create deployment design2 --image=nginx
```

```
deployment.apps/design2 created
```

2. View the wide **kubectl get** output for the `design2` deployment and make note of the `SELECTOR`

```
student@ckad-1:~$ kubectl get deployments.apps design2 -o wide
```

```
NAME     READY  UP-TO-DATE  AVAILABLE  AGE    CONTAINERS  IMAGES  SELECTOR
design2  1/1    1           1          2m13s  nginx       nginx   app=design2
```

3. Use the **-l** option to use the selector to list the pods running inside the deployment. There should be only one pod running.

```
student@ckad-1:~$ kubectl get -l app=design2 pod
```

```
NAME                       READY   STATUS    RESTARTS   AGE
design2-766d48574f-5w274   1/1     Running   0          3m1s
```

4. View the pod details in YAML format using the deployment selector. This time use the **–selector** option. Find the pod label in the output. It should match that of the deployment.

```
student@ckad-1:~$ kubectl get --selector app=design2 pod  -o yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    cni.projectcalico.org/podIP: 192.168.113.222/32
  creationTimestamp: "2020-01-31T16:29:37Z"
  generateName: design2-766d48574f-
  labels:
    app: design2
    pod-template-hash: 766d48574f
....
```

5. Edit the pod label to be your favorite color.

```
student@ckad-1:~$ kubectl edit pod design2-766d48574f-5w274
```

```
....
  labels:
    app: orange                       #<<-- Edit this line
    pod-template-hash: 766d48574f
  name: design2-766d48574f-5w274
....
```

6. Now view how many pods are in the deployment. Then how many have `design2` in their name. Note the `AGE` of the pods.

```
student@ckad-1:~$ kubectl get deployments.apps design2 -o wide
```

```
NAME     READY  UP-TO-DATE  AVAILABLE  AGE  CONTAINERS  IMAGES  SELECTOR
design2  1/1    1           1          56s  nginx       nginx   app=design2
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
student@ckad-1:~$ kubectl get pods | grep design2
```

```
design2-766d48574f-5w274   1/1     Running            0          82s
design2-766d48574f-xttgg   1/1     Running            0          2m12s
```

7. Delete the `design2` deployment.

```
student@ckad-1:~$ kubectl delete deploy design2
```

```
deployment.apps "design2" deleted
```

8. Check again for pods with `design2` in their names. You should find one pod, with an `AGE` of when you first created the deployment. Once the label was edited the deployment created a new pod in order that the status matches the spec and there be a replica running with the intended label.

```
student@ckad-1:~$ kubectl get pods | grep design2
```

```
design2-766d48574f-5w274  1/1     Running            0          38m
```

9. Delete the pod using the **-l** and the label you edited to be your favorite color in a previous step. The command details have been omitted. Use previous steps to figure out these commands.

Exercise 4.5: Setting Pod Resource Limits and Requirements

1. Create a new pod running the `vish/stress` image. A YAML `stress.yaml` file has been included in the course tarball.

2. Run the **top** command on the master and worker nodes. You should find a `stress` command consuming the majority of the CPU on one node, the `worker`. Use **ctrl-c** to exit from top. Delete the deployment.

3. Edit the `stress.yaml` file add in the following limits and requests.

```
student@ckad-1:~$
```

```yaml
....
        name: stressmeout
        resources:                  #<<-- Add this and following six lines
          limits:
            cpu: "1"
            memory: "1Gi"
          requests:
            cpu: "0.5"
            memory: "500Mi"
        args:
        - -cpus
....
```

4. Create the deployment again. Check the status of the pod. You should see that it shows an `OOMKilled` status and a growing number of restarts. You may see a status of `Running` if you catch the pod in early in a restart. If you wait long enough you may see `CrashLoopBackOff`.

```
student@ckad-1:~$ kubectl get pod stressmeout-7fbbbcc887-v9kvb
```

```
NAME                           READY   STATUS      RESTARTS   AGE
stressmeout-7fbbbcc887-v9kvb   0/1     OOMKilled   2          32s
```

5. Delete then edit the deployment. Change the `limit:` parameters such that pod is able to run, but not too much extra resources. Try setting the memory limit to exactly what the stress command requests.

   As we allow the pod to run on the master node, this could cause issues, such as the `kube-apiserver` restarting due to lack of resources. We will also add a `nodeSelector` to use the built in label of `kubernetes.io/hostname`.

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
student@ckad-1:˜$ kubectl delete -f stress.yaml
```

```
student@ckad-1:˜$ vim stress.yaml
```

```
....
    spec:
      nodeSelector:                     #<-- Uncomment and edit
        kubernetes.io/hostname: ckad-2  #<-- to by YOUR worker hostname
      containers:

....
        resources:
          limits:
            cpu: "2"
            memory: "2Gi"
          requests:
....
```

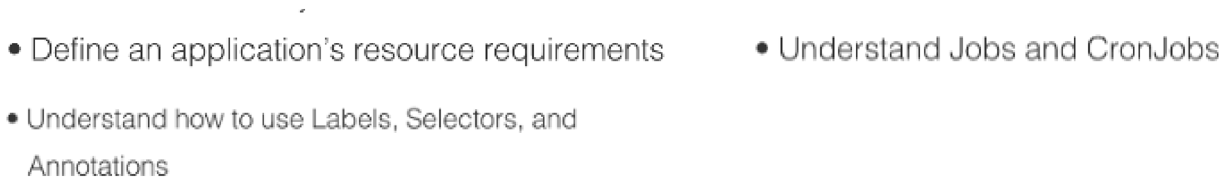
6. Create the deployment and ensure the pod runs without error. Use **top** to verify the stress command is running on one of the nodes and view the pod details to ensure the CPU and memory limits are in use. Also use the **kubectl describe node** command to view the resources your master and worker node are using. The command details have been omitted. Use previous steps to figure out the commands.

Exercise 4.6: Domain Review

Very Important

The source pages and content in this review could change at any time. **IT IS YOUR RESPONSIBILITY TO CHECK THE CURRENT INFORMATION**.

Revisit the CKAD domain list on Curriculum Overview and locate some of the topics we have covered in this chapter. They may be in multiple sections. The graphic below shows the topics covered in this chapter.



- Define an application's resource requirements
- Understand Jobs and CronJobs
- Understand how to use Labels, Selectors, and Annotations

Figure 4.1: **Multiple Domain**

Focus on ensuing you have all the necessary files and processes understood first. Repeat the review until you are sure you have bookmarks of necessary YAML samples and can complete each step quickly, and ensure each object is running properly.

1. Find and use the `design-review1.yaml` file to create a pod.
2. Determine the CPU and memory resource requirements of `design-pod1`.
3. Edit the pod resource requirements such that the CPU limit is exactly twice the amount requested by the container. (Hint: subtract .22)
4. Increase the memory resource limit of the pod until the pod shows a `Running` status. This may require multiple edits and attempts. Determine the minimum amount necessary for the `Running` status to persist at least a minute.

© Copyright the Linux Foundation 2020. All rights reserved.

5. Use the `design-review2.yaml` file to create several pods with various labels.

6. Using **only** the –selector value `tux` to delete only those pods. This should be half of the pods. Hint, you will need to view pod settings to determine the key value as well.

7. Create a new cronjob which runs `busybox` and the `sleep 30` command. Have the cronjob run every three minutes. View the job status to check your work. Change the settings so the pod runs 10 minutes from the current time, every week. For example, if the current time was 2:14PM, I would configure the job to run at 2:24PM, every Monday.

8. Delete any objects created during this review. You may want to delete all but the cronjob if you'd like to see if it runs in 10 minutes. Then delete that object as well.

Ghazela Technology
Academy

Copyright © 2020   adresse : 23, rue el Yakoute ,cité la Ghazelle ,Ariana
MF :1469595/X/A/M/000 – RC : B01912050
Contact : contact at ghazelatc.com
Téléphone :+216 71 866 142 , +216 27 862 155, +216 54 828 018

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

Ghazela Technology
Academy

Copyright © 2020 adresse : 23, rue el Yakoute ,cité la Ghazelle ,Ariana
MF :1469595/X/A/M/000 – RC : B01912050
Contact : contact at ghazelatc.com
Téléphone :+216 71 866 142 , +216 27 862 155, +216 54 828 018

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

Chapter 5

Deployment Configuration

Exercise 5.1: Configure the Deployment: Secrets and ConfigMap

Very Important

Save a copy of your `$HOME/app1/simpleapp.yaml` file, in case you would like to repeat portions of the labs, or you find your file difficult to use due to typos and whitespace issues.

```
student@ckad-1:~$ cp $HOME/app1/simpleapp.yaml $HOME/beforeLab5.yaml
```

Overview

In this lab we will add resources to our deployment with further configuration you may need for production.

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

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

```
student@ckad-1:~/app1$ cd

student@ckad-1:~$ mkdir primary
student@ckad-1:~$ echo c > primary/cyan
student@ckad-1:~$ echo m > primary/magenta
student@ckad-1:~$ echo y > primary/yellow
student@ckad-1:~$ echo k > primary/black
student@ckad-1:~$ echo "known as key" >> primary/black
student@ckad-1:~$ echo blue > favorite
```

2. Generate a **configMap** using each of the three methods.

```
student@ckad-1:~$ kubectl create configmap colors \
  --from-literal=text=black \
  --from-file=./favorite \
  --from-file=./primary/
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
configmap/colors created
```

3. View the newly created **configMap**. Note the way the ingested data is presented.

```
student@ckad-1:~$ kubectl get configmap colors
```

```
NAME      DATA      AGE
colors    6         11s
```

```
student@ckad-1:~$ 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
metadata:
<output_omitted>
```

4. Update the YAML file of the application to make use of the **configMap** as an environmental parameter. Add the six lines from the `env: line` to `key:favorite`.

```
student@ckad-1:~$ vim $HOME/app1/simpleapp.yaml
```

simpleapp.yaml

```
....
    spec:
      containers:
      - image: 10.105.119.236:5000/simpleapp
        env:                                #<-- Add from here
        - name: ilike
          valueFrom:
            configMapKeyRef:
              name: colors
              key: favorite                 #<-- to here
        imagePullPolicy: Always
....
```

5. Delete and re-create the deployment with the new parameters.

```
student@ckad-1-lab-7xtx:~$ kubectl delete deployment try1
```

```
deployment.apps "try1" deleted
```

```
student@ckad-1-lab-7xtx:~$ kubectl create -f $HOME/app1/simpleapp.yaml
```

© Copyright the Linux Foundation 2020. All rights reserved.

```
deployment.apps/try1 created
```

6. Even though the `try1` pod is not in a fully ready state, it is running and useful. Use **kubectl exec** to view a variable's value. View the pod state then verify you can see the `ilike` value within the `simpleapp` container. Note that the use of double dash (- -) tells the shell to pass the following as standard in.

```
student@ckad-1:~$ kubectl get pod
```

```
<output_omitted>
```

```
student@ckad-1:~$ kubectl exec -c simpleapp -it try1-5db9bc6f85-whxbf \
    -- /bin/bash -c 'echo $ilike'
```

```
blue
```

7. Edit the YAML file again, this time adding the another method of using a **configMap**. Edit the file to add three lines. `envFrom` should be indented the same amount as `env` earlier in the file, and `configMapRef` should be indented the same as `configMapKeyRef`.

```
student@ckad-1:~$ vim $HOME/app1/simpleapp.yaml
```

simpleapp.yaml

```
....
           configMapKeyRef:
              name: colors
              key: favorite
        envFrom:              #<-- Add this and the following two lines
        - configMapRef:
            name: colors
        imagePullPolicy: Always
....
```

8. Again delete and recreate the deployment. Check the pods restart.

```
student@ckad-1:~$ kubectl delete deployment try1
```

```
deployment.apps "try1" deleted
```

```
student@ckad-1:~$ kubectl create -f $HOME/app1/simpleapp.yaml
```

```
deployment.apps/try1 created
```

```
student@ckad-1:~$ kubectl get pods
```

```
NAME                      READY  STATUS       RESTARTS   AGE
nginx-6b58d9cdfd-9fnl4    1/1    Running      1          23h
registry-795c6c8b8f-hl5w  1/1    Running      2          23h
try1-d4fbf76fd-46pkb      1/2    Running      0          40s
try1-d4fbf76fd-9kw24      1/2    Running      0          39s
try1-d4fbf76fd-bx9j9      1/2    Running      0          39s
try1-d4fbf76fd-jw8g7      1/2    Running      0          40s
try1-d4fbf76fd-lppl5      1/2    Running      0          39s
try1-d4fbf76fd-xtfd4      1/2    Running      0          40s
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

9. View the settings inside the `try1` container of a pod. The following output is truncated in a few places. Omit the container name to observe the behavior. Also execute a command to see all environmental variables instead of logging into the container first.

```
student@ckad-1:~$ kubectl exec -it try1-d4fbf76fd-46pkb -- /bin/bash -c 'env'
```

```
Defaulting container name to simpleapp.
Use 'kubectl describe pod/try1-d4fbf76fd-46pkb -n default' to see all of the containers in this pod.
REGISTRY_PORT_5000_TCP_ADDR=10.105.119.236
HOSTNAME=try1-d4fbf76fd-46pkb
TERM=xterm
yellow=y
<output_omitted>
REGISTRY_SERVICE_HOST=10.105.119.236
KUBERNETES_SERVICE_PORT=443
REGISTRY_PORT_5000_TCP=tcp://10.105.119.236:5000
KUBERNETES_SERVICE_HOST=10.96.0.1
text=black
REGISTRY_SERVICE_PORT_5000=5000
<output_omitted>
black=k
known as key

<output_omitted>
ilike=blue
<output_omitted>
magenta=m

cyan=c
<output_omitted>
```

10. For greater flexibility and scalability **ConfigMaps** can be created from a YAML file, then deployed and redeployed as necessary. Once ingested into the cluster the data can be retrieved in the same manner as any other object. Create another **configMap**, this time from a YAML file.

```
student@ckad-1:~$ 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
```

```
student@ckad-1:~$ kubectl create -f car-map.yaml
```

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

11. View the ingested data, note that the output is just as in file created.

```
student@ckad-1:~$ kubectl get configmap fast-car -o yaml
```

```
apiVersion: v1
data:
  car.make: Ford
  car.model: Mustang
```

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
  car.trim: Shelby
kind: ConfigMap
metadata:
<output_omitted>
```

12. Add the **configMap** settings to the `simpleapp.yaml` file as a volume. Both containers in the try1 deployment can access to the same volume, using `volumeMounts` statements. Remember that the volume stanza is of equal depth to the containers stanza, and should come after the containers have been declared, the example below has the volume added just before the `status:` output..

```
student@ckad-1:~$ vim $HOME/app1/simpleapp.yaml
```

simpleapp.yaml

```
....
    spec:
      containers:
      - image: 10.105.119.236:5000/simpleapp
        volumeMounts:             #<-- Add this and following two lines
        - mountPath: /etc/cars
          name: car-vol
        env:
        - name: ilike
....
      securityContext: {}
      terminationGracePeriodSeconds: 30
      volumes:                    #<-- Add this and following four lines
      - name: car-vol
        configMap:
          defaultMode: 420
          name: fast-car
status:
....
```

13. Delete and recreate the deployment.

```
student@ckad-1:~$ kubectl delete deployment try1
```

```
deployment.apps "try1" deleted
```

```
student@ckad-1:~$ kubectl create -f $HOME/app1/simpleapp.yaml
```

```
deployment.apps/try1 created
```

14. Verify the deployment is running. Note that we still have not automated the creation of the `/tmp/healthy` file inside the container, as a result the `AVAILABLE` count remains zero until we use the **for** loop to create the file. We will remedy this in the next step.

```
student@ckad-1:~$ kubectl get deployment
```

```
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
nginx       1/1     1            1           1d
registry    1/1     1            1           1d
try1        0/6     6            0           39s
```

15. Our health check was the successful execution of a command. We will edit the command of the existing `readinessProbe` to check for the existence of the mounted configMap file and re-create the deployment. After a minute both containers should become available for each pod in the deployment.

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
student@ckad-1:~$ kubectl delete deployment try1
```

```
deployment.apps "try1" deleted
```

```
student@ckad-1:~$ vim $HOME/app1/simpleapp.yaml
```

simpleapp.yaml

```
....
        readinessProbe:
          exec:
            command:
            - ls                        #<-- Add/Edit this and following line.
            - /etc/cars
          periodSeconds: 5
....
```

```
student@ckad-1:~$ kubectl create -f $HOME/app1/simpleapp.yaml
```

```
deployment.apps/try1 created
```

16. Wait about a minute and view the deployment and pods. All six replicas should be running and report that 2/2 containers are in a ready state within.

```
student@ckad-1:~$ kubectl get deployment
```

```
NAME       READY   UP-TO-DATE   AVAILABLE   AGE
nginx      1/1     1            1           1d
registry   1/1     1            1           1d
try1       6/6     6            6           1m
```

```
student@ckad-1:~$ kubectl get pods
```

```
NAME                          READY     STATUS    RESTARTS    AGE
nginx-6b58d9cdfd-9fnl4        1/1       Running   1           1d
registry-795c6c8b8f-hl5wf     1/1       Running   2           1d
try1-7865dcb948-2dzc8         2/2       Running   0           1m
try1-7865dcb948-7fkh7         2/2       Running   0           1m
try1-7865dcb948-d85bc         2/2       Running   0           1m
try1-7865dcb948-djrcj         2/2       Running   0           1m
try1-7865dcb948-kwlv8         2/2       Running   0           1m
try1-7865dcb948-stb2n         2/2       Running   0           1m
```

17. View a file within the new volume mounted in a container. It should match the data we created inside the configMap. Because the file did not have a carriage-return it will appear prior to the following prompt.

```
student@ckad-1:~$ kubectl exec -c simpleapp -it try1-7865dcb948-stb2n \
    -- /bin/bash -c 'cat /etc/cars/car.trim'
```

```
Shelby student@ckad-1:~$
```

Exercise 5.2: Configure the Deployment: Attaching Storage

There are several types of storage which can be accessed with Kubernetes, with flexibility of storage being essential to scalability. In this exercise we will configure an NFS server. With the NFS server we will create a new **persistent volume (pv)** and a **persistent volume claim (pvc)** to use it.

1. Search for pv and pvc YAML example files on http://kubernetes.io/docs and http://kubernetes.io/blog.

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

2. Use the CreateNFS.sh script from the tarball to set up NFS on your master node. This script will configure the server, export /opt/sfw and create a file /opt/sfw/hello.txt. Use the **find** command to locate the file if you don't remember where you extracted the tar file. This example narrows the search to your $HOME directory. Change for your environment. directory. You may find the same file in more than one sub-directory of the tarfile.

```
student@ckad-1:~$ find $HOME -name CreateNFS.sh
```

```
/home/student/LFD259/SOLUTIONS/s_05/CreateNFS.sh
```

```
student@ckad-1:~$ cp /home/student/LFD259/SOLUTIONS/s_05/CreateNFS.sh $HOME
```

```
student@ckad-1:~$ bash $HOME/CreateNFS.sh
```

```
Hit:1 http://us-central1.gce.archive.ubuntu.com/ubuntu xenial InRelease
Get:2 http://us-central1.gce.archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]

<output_omitted>

Should be ready. Test here and second node

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

3. Test by mounting the resource from your **second node**. Begin by installing the client software.

```
student@ckad-2:~$ sudo apt-get -y install nfs-common nfs-kernel-server
```

```
<output_omitted>
```

4. Test you can see the exported directory using **showmount** from you second node.

```
student@ckad-2:~$ showmount -e ckad-1 #<-- Edit to be first node's name or IP
```

```
Export list for ckad-1:
/opt/sfw *
```

5. Mount the directory. Be aware that unless you edit /etc/fstab this is not a persistent mount. Change out the node name for that of your master node.

```
student@ckad-2:~$ sudo mount ckad-1:/opt/sfw /mnt
```

6. Verify the hello.txt file created by the script can be viewed.

```
student@ckad-2:~$ ls -l /mnt
```

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

7. Return to the master node and create a YAML file for an object with kind **PersistentVolume**. The included example file needs an edit to the server: parameter. 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 incorrect resource will not start. Note that the accessModes do not currently affect actual access and are typically used as labels instead.

```
student@ckad-1:~$ find $HOME -name PVol.yaml
```

```
/home/student/LFD259/SOLUTIONS/s_05/PVol.yaml
```

```
student@ckad-1:~$ cp /home/student/LFD259/SOLUTIONS/s_05/PVol.yaml $HOME
```

```
student@ckad-1:~$ vim PVol.yaml
```

© Copyright the Linux Foundation 2020. All rights reserved.

PVol.yaml

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

8. Create and verify you have a new 1Gi volume named **pvvol-1**. Note the status shows as `Available`. Remember we made two persistent volumes for the image registry earlier.

```
student@ckad-1:~$ kubectl create -f PVol.yaml
```

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

```
student@ckad-1:~$ kubectl get pv
```

```
NAME             CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS  REASON  AGE
pvvol-1          1Gi        RWX            Retain           Available                                  4s
registryvm       200Mi      RWO            Retain           Bound       default/nginx-claim0           4d
task-pv-volume   200Mi      RWO            Retain           Bound       default/registry-claim0        4d
```

9. Now that we have a new volume we will use a **persistent volume claim (pvc)** to use it in a Pod. We should have two existing claims from our local registry.

```
student@ckad-1:~/$ kubectl get pvc
```

```
NAME              STATUS    VOLUME           CAPACITY   ACCESS MODES   STORAGECLASS   AGE
nginx-claim0      Bound     registryvm       200Mi      RWO                           4d
registry-claim0   Bound     task-pv-volume   200Mi      RWO                           4d
```

10. Create or copy a yaml file with the kind **PersistentVolumeClaim**.

```
student@ckad-1:~$ vim pvc.yaml
```

pvc.yaml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-one
spec:
  accessModes:
  - ReadWriteMany
  resources:
     requests:
       storage: 200Mi
```

11. Create and verify the new pvc status is bound. Note the size is 1Gi, even though 200Mi was suggested. Only a volume of at least that size could be used, the first volume with found with at least that much space was chosen.

```
student@ckad-1:~$ kubectl create -f pvc.yaml
```

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

```
student@ckad-1:~$ kubectl get pvc
```

```
NAME          STATUS    VOLUME       CAPACITY   ACCESS MODES   STORAGECLASS   AGE
nginx-claim0      Bound     registryvm       200Mi      RWO                           4d
pvc-one           Bound     pvvol-1          1Gi        RWX                           4s
registry-claim0   Bound     task-pv-volume   200Mi      RWO                           4d
```

12. Now look at the status of the physical volume. It should also show as bound.

```
student@ckad-1:~$ kubectl get pv
```

```
NAME           CAPACITY ACCESS MODES RECLAIM POLICY STATUS
 CLAIM      STORAGECLASS   REASON    AGE
pvvol-1        1Gi      RWX          Retain         Bound
 default/pvc-one                     14m
registryvm     200Mi    RWO          Retain         Bound
 default/nginx-claim0                4d
task-pv-volume 200Mi    RWO          Retain         Bound
 default/registry-claim0             4d
```

13. Edit the simpleapp.yaml file to include two new sections. One section for the container while will use the volume mount point, you should have an existing entry for car-vol. The other section adds a volume to the deployment in general, which you can put after the configMap volume section.

```
student@ckad-1:~$ vim $HOME/app1/simpleapp.yaml
```

simpleapp.yaml

```
....
        volumeMounts:
        - name: car-vol
          mountPath: /etc/cars
        - name: nfs-vol                   #<-- Add this and following line
          mountPath: /opt
....
     volumes:
      - name: car-vol
        configMap:
          defaultMode: 420
          name: fast-car
      - name: nfs-vol                     #<-- Add this and following two lines
        persistentVolumeClaim:
          claimName: pvc-one
status:
....
```

14. Delete and re-create the deployment.

```
student@ckad-1:~$ kubectl delete deployment try1 ; kubectl create -f $HOME/app1/simpleapp.yaml
```

```
deployment.apps "try1" deleted
deployment.apps/try1 created
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

15. View the details any of the pods in the deployment, you should see `nfs-vol` mounted under `/opt`. The use to command line completion with the **tab** key can be helpful for using a pod name.

```
student@ckad-1:~$ kubectl describe pod try1-594fbb5fc7-5k7sj
```

```
<output_omitted>
    Mounts:
      /etc/cars from car-vol (rw)
      /opt from nfs-vol (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-j7cqd (ro)
<output_omitted>
```

Exercise 5.3: Using ConfigMaps Configure Ambassador Containers

In an earlier lab we added a second `Ambassador` container to handle logging. Now that we have learned about using `ConfigMaps` and attaching storage we will use configure our `basic` pod.

1. Review the YAML for our earlier simple pod. Recall that we added an Ambassador style logging container to the pod but had not fully configured the logging.

```
student@ckad-1:~$ cat basic.yaml
```

```
<output_omitted>
  containers:
  - name: webcont
    image: nginx
    ports:
    - containerPort: 80
  - name: fdlogger
    image: fluent/fluentd
```

2. Let us begin by adding shared storage to each container. We will use the `hostPath` storage class to provide the `PV` and `PVC`. First we create the directory.

```
student@ckad-1:~$ sudo mkdir /tmp/weblog
```

3. Now we create a new `PV` to use that directory for the `hostPath` storage class. We will use the `storageClassName` of manual so that only `PVC`s which use that name will bind the resource.

```
student@ckad-1:~$ vim weblog-pv.yaml
```

weblog-pv.yaml

```yaml
kind: PersistentVolume
apiVersion: v1
metadata:
  name: weblog-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 100Mi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/tmp/weblog"
```

4. Create and verify the new PV exists and shows an Available status.

```
student@ckad-1:~$ kubectl create -f weblog-pv.yaml
```

```
persistentvolume/weblog-pv-volume created
```

```
student@ckad-1:~$ kubectl get pv weblog-pv-volume
```

```
NAME             CAPACITY ACCESS MODES  RECLAIM POLICY
  STATUS        CLAIM    STORAGECLASS   REASON    AGE

weblog-pv-volume 100Mi     RWO          Retain
  Available             manual                  21s
```

5. Next we will create a PVC to use the PV we just created.

```
student@ckad-1:~$ vim weblog-pvc.yaml
```

weblog-pvc.yaml

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: weblog-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
```

6. Create the PVC and verify it shows as Bound to the the PV we previously created.

```
student@ckad-1:~$ kubectl create -f weblog-pvc.yaml
```

```
persistentvolumeclaim/weblog-pv-claim created
```

```
student@ckad-1:~$ kubectl get pvc weblog-pv-claim
```

```
NAME             STATUS   VOLUME           CAPACITY ACCESS MODES
   STORAGECLASS    AGE
weblog-pv-claim  Bound    weblog-pv-volume 100Mi    RWO
   manual          79s
```

7. We are ready to add the storage to our pod. We will edit three sections. The first will declare the storage to the pod in general, then two more sections which tell each container where to make the volume available.

```
student@ckad-1:~$ vim basic.yaml
```

basic.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: basicpod
  labels:
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```yaml
    type: webserver
spec:
  volumes:                            #<-- Add three lines, same depth as containers
    - name: weblog-pv-storage
      persistentVolumeClaim:
        claimName: weblog-pv-claim
  containers:
  - name: webcont
    image: nginx
    ports:
    - containerPort: 80
    volumeMounts:                     #<-- Add three lines, same depth as ports
      - mountPath: "/var/log/nginx/"
        name: weblog-pv-storage       # Must match volume name above
  - name: fdlogger
    image: fluent/fluentd
    volumeMounts:                     #<-- Add three lines, same depth as image:
      - mountPath: "/var/log"
        name: weblog-pv-storage       # Must match volume name above
```

8. At this point we can create the pod again. When we create a shell we will find that the `access.log` for **nginx** is no longer a symbolic link pointing to `stdout` it is a writable, zero length file. Leave a **tailf** of the log file running.

```
student@ckad-1:~$ kubectl create -f basic.yaml
```

```
pod/basicpod created
```

```
student@ckad-1:~$ kubectl exec -c webcont -it basicpod -- /bin/bash
```

On Container

```
root@basicpod:/#  ls -l /var/log/nginx/access.log
```

```
-rw-r--r-- 1 root root 0 Oct 18 16:12 /var/log/nginx/access.log
```

```
root@basicpod:/# tail -f /var/log/nginx/access.log
```

9. Open a second connection to your master node. We will use the pod IP as we have not yet configured a `service` to expose the pod.

```
student@ckad-1:~$ kubectl get pods -o wide
```

```
NAME     READY STATUS   RESTARTS  AGE    IP               NODE
   NOMINATED NODE
basicpod 2/2   Running  0         3m26s  192.168.213.181  ckad-1
   <none>
```

10. Use **curl** to view the welcome page of the webserver. When the command completes you should see a new entry added to the log. Right after the `GET` we see a `200` response indicating success. You can use **ctrl-c** and **exit** to return to the host shell prompt.

```
student@ckad-1:~$ curl http://192.168.213.181
```

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

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

On Container

```
192.168.32.128 - - [18/Oct/2018:16:16:21 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.47.0" "-"
```

11. Now that we know the `webcont` container is writing to the `PV` we will configure the logger to use that directory as a source. For greater flexibility we will configure **fluentd** using a `configMap`.

    Fluentd has many options for input and output of data. We will read from a file of the `webcont` container and write to standard out of the `fdlogger` container. The details of the data settings can be found in **fluentd** documentation here: https://docs.fluentd.org/v1.0/categories/config-file

```
student@ckad-1:˜$ vim weblog-configmap.yaml
```

weblog-configmap.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: fluentd-config
  namespace: default
data:
  fluentd.conf: |
    <source>
      @type tail
      format none
      path /var/log/access.log
      tag count.format1
    </source>

    <match *.**>
    @type stdout
    id stdout_output
    </match>
```

12. Create the new configMap.

```
student@ckad-1:˜$ kubectl create -f weblog-configmap.yaml
```

```
configmap/fluentd-config created
```

13. View the logs for both containers in the `basicpod`. You should see some startup information, but not the HTTP traffic.

```
student@ckad-1:˜$ kubectl logs basicpod webcont
```

```
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
```

```
student@ckad-1:˜$  kubectl logs basicpod fdlogger
```

```
2020-09-02 19:32:59 +0000 [info]: reading config file path="/etc/fluentd-config/fluentd.conf"
2020-09-02 19:32:59 +0000 [info]: starting fluentd-0.12.29
2020-09-02 19:32:59 +0000 [info]: gem 'fluent-mixin-config-placeholders' version '0.4.0'
2020-09-02 19:32:59 +0000 [info]: gem 'fluent-mixin-plaintextformatter' version '0.2.6'

<output_omitted>
```

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```

  <source>
    @type tail
    format none
    path /var/log/access.log
<output_omitted>
```

14. Now we will edit the pod yaml file so that the **fluentd** container will mount the configmap as a volume and reference the variables inside the config file. You will add three areas, the volume declaration to the pod, the `env` parameter and the mounting of the volume to the fluentd container

```
student@ckad-1:~$ vim basic.yaml
```

basic.yaml

```
....
  volumes:
    - name: weblog-pv-storage
      persistentVolumeClaim:
        claimName: weblog-pv-claim
    - name: log-config                    #<-- This and two lines following
      configMap:
        name: fluentd-config              # Must match existing configMap
....
    image: fluent/fluentd
    env:                                  #<-- This and two lines following
    - name: FLUENTD_OPT
      value: -c /etc/fluentd-config/fluentd.conf
....
    volumeMounts:
      - mountPath: "/var/log"
        name: weblog-pv-storage
      - name: log-config                  #<-- This and next line
        mountPath: "/etc/fluentd-config"
```

15. At this point we can delete and re-create the pod, which would cause the configmap to be used by the new pod, among other changes.

```
student@ckad-1:~$ kubectl delete pod basicpod
```

```
pod "basicpod" deleted
```

```
student@ckad-1:~$ kubectl create -f basic.yaml
```

```
pod/basicpod created
```

```
student@ckad-1:~$ kubectl get pod basicpod -o wide
```

```
NAME       READY   STATUS    RESTARTS   AGE   IP                NODE     NOMINATED....
basicpod   2/2     Running   0          8s    192.168.171.122   ckad-2   <none>   ....
```

16. Use **curl** a few times to look at the default page served by `basicpod`

```
student@ckad-1:~$ curl http://192.168.171.122
```

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

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

17. Look at the logs for both containers. In addition to the standard startup information, you should also see the HTTP requests from the curl commands you just used at the end of the fdlogger output.

```
student@ckad-1:~$ kubectl logs basicpod webcont
```

```
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
```

```
student@ckad-1:~$  kubectl logs basicpod fdlogger
```

```
2020-09-02 19:32:59 +0000 [info]: reading config file path="/etc/fluentd-config/fluentd.conf"
2020-09-02 19:32:59 +0000 [info]: starting fluentd-0.12.29
2020-09-02 19:32:59 +0000 [info]: gem 'fluent-mixin-config-placeholders' version '0.4.0'
2020-09-02 19:32:59 +0000 [info]: gem 'fluent-mixin-plaintextformatter' version '0.2.6'

<output_omitted>

  <source>
    @type tail
    format none
    path /var/log/access.log

<output_omitted>

2020-09-02 19:47:38 +0000 count.format1: {"message":"192.168.219.64 - - [02/Sep/2020:19:47:38 +0000] \"GET / HTTP
2020-09-02 19:47:41 +0000 count.format1: {"message":"192.168.219.64 - - [02/Sep/2020:19:47:41 +0000] \"GET / HTTP
2020-09-02 19:47:47 +0000 count.format1: {"message":"192.168.219.64 - - [02/Sep/2020:19:47:47 +0000] \"GET / HTTP
```

Exercise 5.4: Rolling Updates and Rollbacks

When we started working with simpleapp we used a **Docker** tag called latest. While this is the default tag when pulling an image, and commonly used, it remains just a string, it may not be the actual latest version of the image.

1. Make a slight change to our source and create a new image. We will use updates and rollbacks with our application. Adding a comment to the last line should be enough for a new image to be generated.

```
student@ckad-1:~$ cd ~/app1

student@ckad-1:~/app1$ vim simple.py

<output_omitted>
## Sleep for five seconds then continue the loop

  time.sleep(5)

## Adding a new comment so image is different.
```

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

2. Build the image again. A new container and image will be created. Verify when successful. There should be a different image ID and a recent creation time.

```
student@ckad-1:~/app1$ sudo docker build -t simpleapp .
```

```
Sending build context to Docker daemon 7.168 kB
Step 1/3 : FROM python:2
 ---> 2863c80c418c
Step 2/3 : ADD simple.py /
 ---> cde8ecf8492b
Removing intermediate container 3e908b76b5b4
Step 3/3 : CMD python ./simple.py
 ---> Running in 354620c97bf5
 ---> cc6bba0ea213
Removing intermediate container 354620c97bf5
Successfully built cc6bba0ea213
```

```
student@ckad-1:~/app1$ sudo docker images
```

```
REPOSITORY                              TAG
 IMAGE ID            CREATED       SIZE
simpleapp                               latest
 cc6bba0ea213    8 seconds ago     886 MB
10.105.119.236:5000/simpleapp           latest
15b5ad19d313     4 days ago        886 MB
<output_omitted>
```

3. Tag and push the updated image to your locally hosted registry. A reminder your IP address will be different than the example below. Use the tag `v2` this time instead of `latest`.

```
student@ckad-1:~/app1$ sudo docker tag simpleapp \
   10.105.119.236:5000/simpleapp:v2

student@ckad-1:~/app1$ sudo docker push 10.105.119.236:5000/simpleapp:v2
```

```
The push refers to a repository [10.105.119.236:5000/simpleapp]
d6153c8cc7c3: Pushed
ca82a2274c57: Layer already exists
de2fbb43bd2a: Layer already exists
4e32c2de91a6: Layer already exists
6e1b48dc2ccc: Layer already exists
ff57bdb79ac8: Layer already exists
6e5e20cbf4a7: Layer already exists
86985c679800: Layer already exists
8fad67424c4e: Layer already exists
v2: digest: sha256:6cf74051d09463d89f1531fceb9c44cbf99006f8d9b407
dd91d8f07baeee7e9c size: 2218
```

4. Connect to a terminal running on your second node. Pull the `latest` image, then pull `v2`. Note the latest did not pull the new version of the image. Again, remember to use the IP for your locally hosted registry. You'll note the digest is different.

```
student@ckad-2:~$ sudo docker pull 10.105.119.236:5000/simpleapp
```

```
Using default tag: latest
latest: Pulling from simpleapp
Digest: sha256:cefa3305c36101d32399baf0919d3482ae8a53c926688be33
86f9bbc04e490a5
Status: Image is up to date for 10.105.119.236:5000/simpleapp:latest
```

```
student@ckad-2:~$ sudo docker pull 10.105.119.236:5000/simpleapp:v2
```

```
v2: Pulling from simpleapp
f65523718fc5: Already exists
1d2dd88bf649: Already exists
c09558828658: Already exists
0e1d7c9e6c06: Already exists
c6b6fe164861: Already exists
45097146116f: Already exists
f21f8abae4c4: Already exists
1c39556edcd0: Already exists
fa67749bf47d: Pull complete
Digest: sha256:6cf74051d09463d89f1531fceb9c44cbf99006f8d9b407dd91d8
f07baeee7e9c
Status: Downloaded newer image for 10.105.119.236:5000/simpleapp:v2
```

5. Use **kubectl edit** to update the image for the `try1` deployment to use `v2`. As we are only changing one parameter we could also use the **kubectl set** command. Note that the configuration file has not been updated, so a delete or a replace command would not include the new version. It can take the pods up to a minute to delete and to recreate each pod in sequence.

```
student@ckad-1:~/app1$ kubectl edit deployment try1
```

```
....
      containers:
      - image: 10.105.119.236:5000/simpleapp:v2   #<-- Edit tag
        imagePullPolicy: Always
....
```

6. Verify each of the pods has been recreated and is using the new version of the image. Note some messages will show the scaling down of the old **replicaset**, others should show the scaling up using the new image.

```
student@ckad-1:~/app1$ kubectl get events
```

```
42m         Normal    ScalingReplicaSet   Deployment   Scaled up replica set try1-7fdbb5d557 to 6
32s         Normal    ScalingReplicaSet   Deployment   Scaled up replica set try1-7fd7459fc6 to 2
32s         Normal    ScalingReplicaSet   Deployment   Scaled down replica set try1-7fdbb5d557 to 5
32s         Normal    ScalingReplicaSet   Deployment   Scaled up replica set try1-7fd7459fc6 to 3
23s         Normal    ScalingReplicaSet   Deployment   Scaled down replica set try1-7fdbb5d557 to 4
23s         Normal    ScalingReplicaSet   Deployment   Scaled up replica set try1-7fd7459fc6 to 4
22s         Normal    ScalingReplicaSet   Deployment   Scaled down replica set try1-7fdbb5d557 to 3
22s         Normal    ScalingReplicaSet   Deployment   Scaled up replica set try1-7fd7459fc6 to 5
18s         Normal    ScalingReplicaSet   Deployment   Scaled down replica set try1-7fdbb5d557 to 2
18s         Normal    ScalingReplicaSet   Deployment   Scaled up replica set try1-7fd7459fc6 to 6
8s          Normal    ScalingReplicaSet   Deployment   (combined from similar events):
Scaled down replica set try1-7fdbb5d557 to 0

```

7. View the images of a Pod in the deployment. Narrow the output to just view the images. The `goproxy` remains unchanged, but the `simpleapp` should now be `v2`.

```
student@ckad-1:~/app1$ kubectl describe pod try1-895fccfb-ttqdn |grep Image
```

```
    Image:          10.105.119.236:5000/simpleapp:v2
    Image ID:\
       docker-pullable://10.105.119.236:5000/simpleapp@sha256:6cf74051d09
463d89f1531fceb9c44cbf99006f8d9b407dd91d8f07baeee7e9c
    Image:          k8s.gcr.io/goproxy:0.1
    Image ID:\
       docker-pullable://k8s.gcr.io/goproxy@sha256:5334c7ad43048e3538775c
b09aaf184f5e8acf4b0ea60e3bc8f1d93c209865a5
```

8. View the update history of the deployment.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
student@ckad-1:~/app1$ kubectl rollout history deployment try1
```

```
deployments "try1"
REVISION  CHANGE-CAUSE
1         <none>
2         <none>
```

9. Compare the output of the **rollout history** for the two revisions. Images and labels should be different, with the image `v2` being the change we made.

```
student@ckad-1:~/app1$ kubectl rollout history deployment try1 --revision=1 > one.out

student@ckad-1:~/app1$ kubectl rollout history deployment try1 --revision=2 > two.out

student@ckad-/app11:~$ diff one.out two.out
```

```
1c1
< deployments "try1" with revision #1
---
> deployments "try1" with revision #2
3c3
<   Labels:        pod-template-hash=1509661973
---
>   Labels:        pod-template-hash=45197796
7c7
<     Image:        10.105.119.236:5000/simpleapp
---
>     Image:        10.105.119.236:5000/simpleapp:v2
```

10. View what would be undone using the **–dry-run** option while undoing the rollout. This allows us to see the new template prior to using it.

```
student@ckad-1:~/app1$ kubectl rollout undo --dry-run=client deployment/try1
```

```
deployment.apps/try1
Pod Template:
  Labels:        pod-template-hash=1509661973
        run=try1
  Containers:
   try1:
    Image:        10.105.119.236:5000/simpleapp:latest
    Port:        <none>
<output_omitted>
```

11. View the pods. Depending on how fast you type the `try1` pods should be about 2 minutes old.

```
student@ckad-1:~/app1$ kubectl get pods
```

```
NAME                           READY     STATUS     RESTARTS     AGE
nginx-6b58d9cdfd-9fnl4         1/1       Running    1            5d
registry-795c6c8b8f-hl5wf      1/1       Running    2            5d
try1-594fbb5fc7-7dl7c          2/2       Running    0            2m
try1-594fbb5fc7-8mxlb          2/2       Running    0            2m
try1-594fbb5fc7-jr7h7          2/2       Running    0            2m
try1-594fbb5fc7-s24wt          2/2       Running    0            2m
try1-594fbb5fc7-xfffg          2/2       Running    0            2m
try1-594fbb5fc7-zfmz8          2/2       Running    0            2m
```

12. In our case there are only two revisions, which is also the default number kept. Were there more we could choose a particular version. The following command would have the same effect as the previous, without the **–dry-run** option.

```
student@ckad-1:~/app1$ kubectl rollout undo deployment try1 --to-revision=1
```

© Copyright the Linux Foundation 2020. All rights reserved.

```
deployment.apps/try1 rolled back
```

13. Again, it can take a bit for the pods to be terminated and re-created. Keep checking back until they are all running again.

```
student@ckad-1:~/app1$ kubectl get pods
```

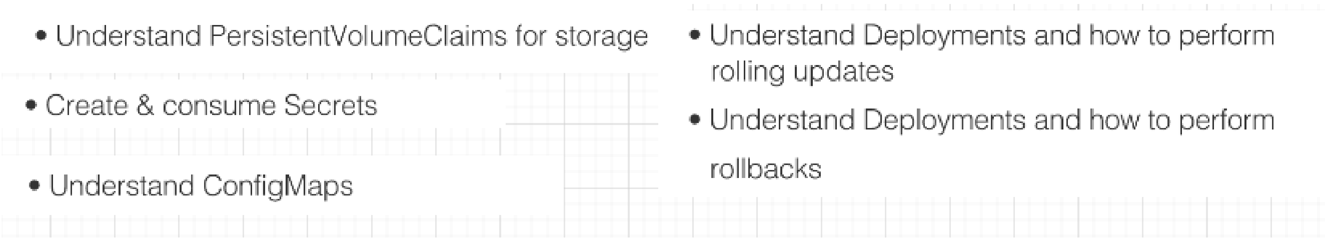
```
NAME                          READY     STATUS        RESTARTS    AGE
nginx-6b58d9cdfd-9fnl4        1/1       Running       1           5d
registry-795c6c8b8f-hl5wf     1/1       Running       2           5d
try1-594fbb5fc7-7dl7c         2/2       Terminating   0           3m
try1-594fbb5fc7-8mxlb         0/2       Terminating   0           2m
try1-594fbb5fc7-jr7h7         2/2       Terminating   0           3m
try1-594fbb5fc7-s24wt         2/2       Terminating   0           2m
try1-594fbb5fc7-xfffg         2/2       Terminating   0           3m
try1-594fbb5fc7-zfmz8         1/2       Terminating   0           2m
try1-895fccfb-8dn4b           2/2       Running       0           22s
try1-895fccfb-kz72j           2/2       Running       0           10s
try1-895fccfb-rxxtw           2/2       Running       0           24s
try1-895fccfb-srwq4           1/2       Running       0           11s
try1-895fccfb-vkvmb           2/2       Running       0           31s
try1-895fccfb-z46qr           2/2       Running       0           31s
```

Exercise 5.5: Domain Review

Very Important

The source pages and content in this review could change at any time. **IT IS YOUR RESPONSIBILITY TO CHECK THE CURRENT INFORMATION.**

Revisit the CKAD domain list on Curriculum Overview and locate some of the topics we have covered in this chapter. The graphic shows bullet points covered in this chapter.



- Understand PersistentVolumeClaims for storage
- Create & consume Secrets
- Understand ConfigMaps
- Understand Deployments and how to perform rolling updates
- Understand Deployments and how to perform rollbacks

Figure 5.1: **Deployment Related Domain Topics**

Focus on ensuing you have all the necessary files and processes understood first. Repeat the review until you are sure you have bookmarks of necessary YAML samples and can complete each step quickly, and ensure each object is running properly.

Using only the allowed browser, URLs, and subdomains search for and bookmark a YAML example to create and configure the resources called for in this review.

1. Create a new secret called `specialofday` using the key `entree` and the value `meatloaf`.
2. Create a new deployment called `foodie` running the `nginx` image.
3. Add the `specialofday` secret to pod mounted as a volume under the `/food/` directory.
4. Execute a bash shell inside a `foodie` pod and verify the secret has been properly mounted.

5. Update the deployment to use the `nginx:1.12.1-alpine` image and verify the new image is in use.
6. Roll back the deployment and verify the typical, current stable version of nginx is in use again.
7. Create a new 200M NFS volume called `reviewvol` using the NFS server configured earlier in the lab.
8. Create a new PVC called `reviewpvc` which will uses the `reviewvol` volume.
9. Edit the deployment to use the PVC and mount the volume under `/newvol`
10. Execute a bash shell into the nginx container and verify the volume has been mounted.
11. Delete any resources created during this review.

Ghazela Technology
Academy

Copyright © 2020 adresse : 23, rue el Yakoute ,cité la Ghazelle ,Ariana
MF :1469595/X/A/M/000 – RC : B01912050
Contact : contact at ghazelatc.com
Téléphone :+216 71 866 142 , +216 27 862 155, +216 54 828 018

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

Chapter 6

Security

Exercise 6.1: Set SecurityContext for a Pod and Container

Working with Security: Overview

In this lab we will implement security features for new applications, as the simpleapp YAML file is getting long and more difficult to read. Kubernetes architecture favors smaller, decoupled, and transient applications working together. We'll continue to emulate that in our exercises.

In this exercise we will create two new applications. One will be limited in its access to the host node, but have access to encoded data. The second will use a `network security policy` to move from the default all-access Kubernetes policies to a mostly closed network. First we will set `security contexts` for pods and containers, then create and consume secrets, then finish with configuring a network security policy.

1. Begin by making a new directory for our second application. Change into that directory.

```
student@ckad-1:~$ mkdir $HOME/app2

student@ckad-1:~$ cd $HOME/app2/
```

2. Create a YAML file for the second application. In the example below we are using a simple image, busybox, which allows access to a shell, but not much more. We will add a `runAsUser` to both the pod as well as the container.

```
student@ckad-1:~/app2$ vim second.yaml
```

second.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secondapp
spec:
  securityContext:
    runAsUser: 1000
  containers:
  - name: busy
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```yaml
    image: busybox
    command:
      - sleep
      - "3600"
    securityContext:
      runAsUser: 2000
      allowPrivilegeEscalation: false
```

3. Create the secondapp pod and verify it's running. Unlike the previous deployment this application is running as a pod. Look at the YAML output, to compare and contrast with what a deployment looks like. The status section probably has the largest contrast.

```
student@ckad-1:~/app2$ kubectl create -f second.yaml
```

```
pod/secondapp created
```

```
student@ckad-1:~/app2$ kubectl get  pod secondapp
```

```
NAME        READY     STATUS    RESTARTS    AGE
secondapp   1/1       Running   0           21s
```

```
student@ckad-1:~/app2$ kubectl get pod secondapp -o yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    cni.projectcalico.org/podIP: 192.168.158.97/32
  creationTimestamp: "2019-11-03T21:23:12Z"
  name: secondapp
<output_omitted>
```

4. Execute a Bourne shell within the Pod. Check the user ID of the shell and other processes. It should show the container setting, not the pod. This allows for multiple containers within a pod to customize their UID if desired. As there is only one container in the pod we do not need to use the **-c busy** option.

```
student@ckad-1:~/app2$ kubectl exec -it secondapp -- sh
```

On Container

```
/ $ ps aux
```

```
PID   USER     TIME  COMMAND
    1   2000        0:00   sleep 3600
    8   2000        0:00   sh
   12   2000       0:00   ps aux
```

5. While here check the capabilities of the kernel. In upcoming steps we will modify these values.

On Container

```
/ $ grep Cap /proc/1/status
```

```
CapInh:         00000000a80425fb
CapPrm:         0000000000000000
```

© Copyright the Linux Foundation 2020. All rights reserved.

```
CapEff:         0000000000000000
CapBnd:         00000000a80425fb
CapAmb:         0000000000000000
```

```
/ $ exit
```

6. Use the capability shell wrapper tool, the **capsh** command, to decode the output. We will view and compare the output in a few steps. Note that there are 14 comma separated capabilities listed.

```
student@ckad-1:~/app2$ capsh --decode=00000000a80425fb
```

```
0x00000000a80425fb=cap_chown,cap_dac_override,cap_fowner,cap_fsetid,
cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_net_bind_service,
cap_net_raw,cap_sys_chroot,cap_mknod,cap_audit_write,cap_setfcap
```

7. Edit the YAML file to include new capabilities for the container. A capability allows granting of specific, elevated privileges without granting full root access. We will be setting **NET_ADMIN** to allow interface, routing, and other network configuration. We'll also set **SYS_TIME**, which allows system clock configuration. More on kernel capabilities can be read here: https://github.com/torvalds/linux/blob/master/include/uapi/linux/capability.h

   It can take up to a minute for the pod to fully terminate, allowing the future pod to be created.

```
student@ckad-1:~/app2$ kubectl delete pod secondapp
```

```
pod "secondapp" deleted
```

```
student@ckad-1:~/app2$ vim second.yaml
```

second.yaml

```
<output_omitted>
      - sleep
      - "3600"
    securityContext:
      runAsUser: 2000
      allowPrivilegeEscalation: false
      capabilities:                      #<-- Add this and following line
        add: ["NET_ADMIN", "SYS_TIME"]
```

8. Create the pod again. Execute a shell within the container and review the Cap settings under /proc/1/status. They should be different from the previous instance.

```
student@ckad-1:~/app2$ kubectl create -f second.yaml
```

```
pod/secondapp created
```

```
student@ckad-1:~/app2$ kubectl exec -it secondapp -- sh
```

On Container

```
/ $ grep Cap /proc/1/status
```

```
CapInh:         00000000aa0435fb
CapPrm:         0000000000000000
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
CapEff:        0000000000000000
CapBnd:        00000000aa0435fb
CapAmb:        0000000000000000
```

```
/ $ exit
```

9. Decode the output again. Note that the instance now has 16 comma delimited capabilities listed. **cap_net_admin** is listed as well as **cap_sys_time**.

```
student@ckad-1:~/app2$ capsh --decode=00000000aa0435fb
```

```
0x00000000aa0435fb=cap_chown,cap_dac_override,cap_fowner,
cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,
cap_net_bind_service,cap_net_admin,cap_net_raw,cap_sys_chroot,
cap_sys_time,cap_mknod,cap_audit_write,cap_setfcap
```

Exercise 6.2: Create and consume Secrets

Secrets are consumed in a manner similar to `ConfigMaps`, covered in an earlier lab. While at-rest encryption is just now enabled, historically a secret was just base64 encoded. There are three types of encryption which can be configured.

1. Begin by generating an encoded password.

```
student@ckad-1:~/app2$ echo LFTr@1n | base64
```

```
TEZUckAxbgo=
```

2. Create a YAML file for the object with an API object kind set to Secret. Use the encoded key as a password parameter.

```
student@ckad-1:~/app2$ vim secret.yaml
```

secret.yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: lfsecret
data:
  password: TEZUckAxbgo=
```

3. Ingest the new object into the cluster.

```
student@ckad-1:~/app2$ kubectl create -f secret.yaml
```

```
secret/lfsecret created
```

4. Edit secondapp YAML file to use the secret as a volume mounted under `/mysqlpassword`. `volumeMounts:` lines up with the container `name:` and `volumes:` lines up with `containers:` Note the pod will restart when the sleep command finishes every 3600 seconds, or every hour.

```
student@ckad-1:~/app2$ vim second.yaml
```

© Copyright the Linux Foundation 2020. All rights reserved.

second.yaml

```
....
      runAsUser: 2000
      allowPrivilegeEscalation: false
      capabilities:
        add: ["NET_ADMIN", "SYS_TIME"]
    volumeMounts:                          #<-- Add this and six following lines
    - name: mysql
      mountPath: /mysqlpassword
  volumes:
  - name: mysql
    secret:
      secretName: lfsecret
```

```
student@ckad-1:~/app2$ kubectl delete pod secondapp
```

```
pod "secondapp" deleted
```

```
student@ckad-1:~/app2$ kubectl create -f second.yaml
```

```
pod/secondapp created
```

5. Verify the pod is running, then check if the password is mounted where expected. We will find that the password is available in its clear-text, decoded state.

```
student@ckad-1:~/app2$ kubectl get pod secondapp
```

```
NAME        READY     STATUS    RESTARTS   AGE
secondapp   1/1       Running   0          34s
```

```
student@ckad-1:~/app2$ kubectl exec -ti secondapp -- /bin/sh
```

On Container

```
/ $ cat /mysqlpassword/password
```

```
LFTr@1n
```

6. View the location of the directory. Note it is a symbolic link to `..data` which is also a symbolic link to another directory. After taking a look at the filesystem within the container, exit back to the node.

On Container

```
/ $ cd /mysqlpassword/
```

```
/mysqlpassword $ ls
```

```
password
```

```
/mysqlpassword $ ls -al
```

```
total 4
drwxrwxrwt    3 root     root           100 Apr 11 07:24 .
drwxr-xr-x   21 root     root          4096 Apr 11 22:30 ..
drwxr-xr-x    2 root     root            60 Apr 11 07:24 ..4984_11_04_07_24_47.831222818
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
lrwxrwxrwx    1 root     root           31 Apr 11 07:24 ..data -> ..4984_11_04_07_24_47.831222818
lrwxrwxrwx    1 root     root           15 Apr 11 07:24 password -> ..data/password
```

```
/mysqlpassword $ exit
```

Exercise 6.3: Working with ServiceAccounts

We can use `ServiceAccounts` to assign cluster roles, or the ability to use particular HTTP verbs. In this section we will create a new `ServiceAccount` and grant it access to view secrets.

1. Begin by viewing secrets, both in the default namespace as well as all.

```
student@ckad-1:~/app2$ cd
```

```
student@ckad-1:~$ kubectl get secrets
```

```
NAME                  TYPE                                  DATA   AGE
default-token-c4rdg   kubernetes.io/service-account-token   3      4d16h
lfsecret              Opaque                                1      6m5s
```

```
student@ckad-1:~$ kubectl get secrets --all-namespaces
```

```
NAMESPACE     NAME
 TYPE                                      DATA   AGE
default       default-token-c4rdg
 kubernetes.io/service-account-token   3      4d16h
kube-public   default-token-zqzbg
 kubernetes.io/service-account-token   3      4d16h
kube-system   attachdetach-controller-token-wxzvc
 kubernetes.io/service-account-token   3      4d16h
<output_omitted>
```

2. We can see that each agent uses a secret in order to interact with the API server. We will create a new `ServiceAccount` which will have access.

```
student@ckad-1:~$ vim serviceaccount.yaml
```

serviceaccount.yaml

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: secret-access-sa
```

```
student@ckad-1:~$ kubectl create -f serviceaccount.yaml
```

```
serviceaccount/secret-access-sa created
```

```
student@ckad-1:~$ kubectl get serviceaccounts
```

```
NAME               SECRETS   AGE
default            1         1d17h
secret-access-sa   1         34s
```

© Copyright the Linux Foundation 2020. All rights reserved.

3. Now we will create a `ClusterRole` which will list the actual actions allowed cluster-wide. We will look at an existing role to see the syntax.

```
student@ckad-1:~$ kubectl get clusterroles
```

```
NAME                                                      AGE
admin                                                     1d17h
calico-cni-plugin                                         1d17h
calico-kube-controllers                                   1d17h
cluster-admin                                             1d17h
<output_omitted>
```

4. View the details for the `admin` and compare it to the `cluster-admin`. The `admin` has particular actions allowed, but `cluster-admin` has the meta-character '*' allowing all actions.

```
student@ckad-1:~$ kubectl get clusterroles admin -o yaml
```

```
<output_omitted>
```

```
student@ckad-1:~$ kubectl get clusterroles cluster-admin -o yaml
```

```
<output_omitted>
```

5. Using some of the output above, we will create our own file.

```
student@ckad-1:~$ vim clusterrole.yaml
```

clusterrole.yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
 name: secret-access-cr
rules:
- apiGroups:
  - ""
  resources:
  - secrets
  verbs:
  - get
  - list
```

6. Create and verify the new `ClusterRole`.

```
student@ckad-1:~$ kubectl create -f clusterrole.yaml
```

```
clusterrole.rbac.authorization.k8s.io/secret-access-cr created
```

```
student@ckad-1:~$ kubectl get clusterrole secret-access-cr -o yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  creationTimestamp: 2018-10-18T19:27:24Z
  name: secret-access-cr
<output_omitted>
```

7. Now we bind the role to the account. Create another YAML file which uses `roleRef:`:

```
student@ckad-1:~$ vim rolebinding.yaml
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

rolebinding.yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
 name: secret-rb
subjects:
- kind: ServiceAccount
  name: secret-access-sa
roleRef:
 kind: ClusterRole
 name: secret-access-cr
 apiGroup: rbac.authorization.k8s.io
```

8. Create the new `RoleBinding` and verify.

```
student@ckad-1:~$ kubectl create -f rolebinding.yaml
```

```
rolebinding.rbac.authorization.k8s.io/secret-rb created
```

```
student@ckad-1:~$ kubectl get rolebindings
```

```
NAME        AGE
secret-rb   17s
```

9. View the `secondapp` pod and **grep** for secret settings. Note that it uses the default settings.

```
student@ckad-1:~$ kubectl describe pod secondapp |grep -i secret
```

```
   /var/run/secrets/kubernetes.io/serviceaccount from
default-token-c4rdg (ro)
   Type:         Secret (a volume populated by a Secret)
   SecretName:   lfsecret
   Type:         Secret (a volume populated by a Secret)
   SecretName:   default-token-c4rdg
```

10. Edit the `second.yaml` file and add the use of the `serviceAccount`.

```
student@ckad-1:~$ vim $HOME/app2/second.yaml
```

second.yaml

```yaml
....
  name: secondapp
spec:
  serviceAccountName: secret-access-sa  #<-- Add this line
  securityContext:
    runAsUser: 1000
....
```

11. We will delete the `secondapp` pod if still running, then create it again. View what the secret is by default.

```
student@ckad-1:~$ kubectl delete pod secondapp ; kubectl create -f $HOME/app2/second.yaml
```

```
pod "secondapp" deleted
pod/secondapp created
```

```
student@ckad-1:~$ kubectl describe pod secondapp | grep -i secret
```

© Copyright the Linux Foundation 2020. All rights reserved.

```
   /var/run/secrets/kubernetes.io/serviceaccount from
secret-access-sa-token-wd7vm (ro)
 secret-access-sa-token-wd7vm:
   Type:         Secret (a volume populated by a Secret)
   SecretName:   secret-access-sa-token-wd7vm

```

Exercise 6.4: Implement a NetworkPolicy

An early architecture decision with Kubernetes was non-isolation, that all pods were able to connect to all other pods and nodes by design. In more recent releases the use of a NetworkPolicy allows for pod isolation. The policy only has effect when the network plugin, like **Project Calico**, are capable of honoring them. If used with a plugin like **flannel** they will have no effect. The use of matchLabels allows for more granular selection within the namespace which can be selected using a namespaceSelector. Using multiple labels can allow for complex application of rules. More information can be found here: https://kubernetes.io/docs/concepts/services-networking/network-policies

1. Begin by creating a default policy which denies all traffic. Once ingested into the cluster this will affect every pod not selected by another policy, creating a mostly-closed environment. If you want to only deny ingress or egress traffic you can remove the other policyType.

```
student@ckad-1:~$ cd $HOME/app2/

student@ckad-1:~/app2$ vim allclosed.yaml
```

allclosed.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-default
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

2. Before we can test the new network policy we need to make sure network access works without it applied. Update **secondapp** to include a new container running **nginx**, then test access. Begin by adding two lines for the **nginx** image and name webserver, as found below. It takes a bit for the pod to terminate, so we'll delete then edit the file.

```
student@ckad-1:~/app2$ kubectl delete pod secondapp
```

```
pod "secondapp" deleted
```

```
student@ckad-1:~/app2$ vim second.yaml
```

second.yaml

```yaml
  runAsUser: 1000
containers:
- name: webserver                   #<-- Add this and following line
  image: nginx
- name: busy
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
    image: busybox
    command:
```

3. Create the new pod. Be aware the pod will move from `ContainerCreating` to Error to `CrashLoopBackOff`, as only one of the containers will start. We will troubleshoot the error in following steps.

```
student@ckad-1:~/app2$ kubectl create -f second.yaml
```

```
pod/secondapp created
```

```
student@ckad-1:~/app2$ kubectl get pods
```

```
NAME                        READY  STATUS            RESTARTS   AGE
nginx-6b58d9cdfd-9fnl4      1/1    Running           1          2d
Registry-795c6c8b8f-hl5wf   1/1    Running           2          2d
secondapp                   1/2    CrashLoopBackOff  1          13s
<output_omitted>
```

4. Take a closer look at the events leading up to the failure. The images were pulled and the container was started. It was the full execution of the container which failed.

```
student@ckad-1:~/app2$ kubectl get event
```

```
<output_omitted>
25s         Normal    Scheduled   Pod    Successfully assigned default/secondapp to ckad-1
4s          Normal    Pulling     Pod    pulling image "nginx"
2s          Normal    Pulled      Pod    Successfully pulled image "nginx"
2s          Normal    Created     Pod    Created container
2s          Normal    Started     Pod    Started container
23s         Normal    Pulling     Pod    pulling image "busybox"
21s         Normal    Pulled      Pod    Successfully pulled image "busybox"
21s         Normal    Created     Pod    Created container
21s         Normal    Started     Pod    Started container
1s          Warning   BackOff     Pod    Back-off restarting failed container
```

5. View the logs of the **webserver** container mentioned in the previous output. Note there are errors about the user directive and not having permission to make directories.

```
student@ckad-1:~/app2$ kubectl logs secondapp webserver
```

```
2018/04/13 19:51:13 [warn] 1#1: the "user" directive makes sense
only if the master process runs with super-user privileges,
 ignored in /etc/nginx/nginx.conf:2
nginx: [warn] the "user" directive makes sense only if the master
process runs with super-user privileges,
 ignored in /etc/nginx/nginx.conf:2
2018/04/13 19:51:13 [emerg] 1#1: mkdir() "/var/cache/nginx/client_temp"
failed (13: Permission denied)
nginx: [emerg] mkdir() "/var/cache/nginx/client_temp" failed
(13: Permission denied)
```

6. Delete the pods. Edit the YAML file to comment out the setting of a UID for the entire pod.

```
student@ckad-1:~/app2$ kubectl delete -f second.yaml
```

```
pod "secondapp" deleted
```

```
student@ckad-1:~/app2$ vim second.yaml
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

second.yaml

```
spec:
  serviceAccountName: secret-access-sa
#  securityContext:                        #<-- Comment this and following line
#    runAsUser: 1000
  containers:
  - name: webserver
```

7. Create the pod again. This time both containers should run. You may have to wait for the previous pod to fully terminate, depending on how fast you type.

```
student@ckad-1:~/app2$ kubectl create -f second.yaml
```

```
pod/secondapp created
```

```
student@ckad-1:~/app2$ kubectl get pods
```

```
NAME                       READY     STATUS      RESTARTS    AGE
<output_omitted>
secondapp                  2/2       Running     0           5s
```

8. Expose the **webserver** using a `NodePort` service. Expect an error due to lack of labels.

```
student@ckad-1:~/app2$ kubectl expose pod secondapp --type=NodePort --port=80
```

```
error: couldn't retrieve selectors via --selector flag or
introspection: the pod has no labels and cannot be exposed
See 'kubectl expose -h' for help and examples.
```

9. Edit the YAML file to add a label in the metadata, adding the `example: second` label right after the pod name. Note you can delete several resources at once by passing the YAML file to the delete command. Delete and recreate the pod. It may take up to a minute for the pod to shut down.

```
student@ckad-1:~/app2$ kubectl delete -f second.yaml
```

```
pod "secondapp" deleted
```

```
student@ckad-1:~/app2$ vim second.yaml
```

second.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: secondapp
  labels:                         #<-- This and following line
    example: second
spec:
#  securityContext:
#    runAsUser: 1000
<output_omitted>
```

```
student@ckad-1:~/app2$ kubectl create -f second.yaml
```

```
pod/secondapp created
```

```
student@ckad-1:~/app2$ kubectl get pods
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
NAME                            READY     STATUS      RESTARTS    AGE
<output_omitted>
secondapp                       2/2       Running     0           15s
```

10. This time we will expose a `NodePort` again, and create the service separately, then add a label to illustrate how labels are essential for tying resources together inside of kubernetes.

```
student@ckad-1:~/app2$ kubectl create service nodeport secondapp --tcp=80
```

```
service/secondapp created
```

11. Look at the details of the service. Note the selector is set to `app: secondapp`. Also take note of the nodePort, which is `31655` in the example below, yours may be different.

```
student@ckad-1:~/app2$ kubectl get svc secondapp -o yaml
```

```
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: "2020-04-16T04:40:07Z"
  labels:
    example: second
  managedFields:

....

spec:
  clusterIP: 10.97.96.75
  externalTrafficPolicy: Cluster
  ports:
  - nodePort: 31665
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    example: second
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
```

12. Test access to the service using **curl** and the `ClusterIP` shown in the previous output. As the label does not match any other resources, the **curl** command should fail. If it hangs **control-c** to exit back to the shell.

```
student@ckad-1:~/app2$ curl http://10.97.96.75
```

13. Edit the service. We will change the label to match **secondapp**, and set the `nodePort` to a new port, one that may have been specifically opened by our firewall team, port `32000`.

```
student@ckad-1:~/app2$ kubectl edit svc secondapp
```

YAML secondapp service

```
<output_omitted>
 ports:
 - name: "80"
   nodePort: 32000       #<-- Edit this line
   port: 80
   protocol: TCP
   targetPort: 80
```

© Copyright the Linux Foundation 2020. All rights reserved.

```
  selector:
    example: second       #<-- Edit this line
  sessionAffinity: None
<output_omitted>
```

14. Verify the updated port number is showing properly, and take note of the ClusterIP. The example below shows a ClusterIP of `10.97.96.75` and a port of `32000` as expected.

```
student@ckad-1:~/app2$ kubectl get svc
```

```
NAME          TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
<output_omitted>
secondapp     NodePort   10.97.96.75     <none>        80:32000/TCP   5m
```

15. Test access to the high port. You should get the default nginx welcome page both if you test from the node to the `ClusterIP:<low-port-number>` and from the exterior `hostIP:<high-port-number>`. As the high port is randomly generated make sure it's available. Both of your nodes should be exposing the web server on port 32000. The example shows the use of the **curl** command, you could also use a web browser.

```
student@ckad-1:~/app2$ curl http://10.97.96.75
```

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

```
[user@laptop ~]$ curl http://35.184.219.5:32000
```

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

16. Now test egress from a container to the outside world. We'll use the **netcat** command to verify access to a running web server on port 80. First test local access to nginx, then a remote server.

```
student@ckad-1:~/app2$ kubectl exec -it -c busy secondapp -- sh
```

On Container

```
/ $ nc -vz 127.0.0.1 80
```

```
127.0.0.1 (127.0.0.1:80) open
```

```
/ $ nc -vz www.linux.com 80
```

```
www.linux.com (151.101.185.5:80) open
```

```
/ $ exit
```

Exercise 6.5: Testing the Policy

1. Now that we have tested both ingress and egress we can implement the network policy.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
student@ckad-1:~/app2$ kubectl create -f $HOME/app2/allclosed.yaml
```

```
networkpolicy.networking.k8s.io/deny-default created
```

2. Use the ingress and egress tests again. Three of the four should eventually timeout. Start by testing from outside the cluster, and interrupt if you get tired of waiting.

```
[user@laptop ~]$ curl http://35.184.219.5:32000
```

```
curl: (7) Failed to connect to 35.184.219.5 port
32000: Connection timed out
```

3. Then test from the host to the container.

```
student@ckad-1:~/app2$ curl http://10.97.96.75:80
```

```
curl: (7) Failed to connect to 10.97.96.75 port 80: Connection timed out
```

4. Now test egress. From container to container should work, as the filter is outside of the pod. Then test egress to an external web page. It should eventually timeout.

```
student@ckad-1:~/app2$ kubectl exec -it -c busy secondapp -- sh
```

On Container

```
/ $ nc -vz 127.0.0.1 80
```

```
127.0.0.1 (127.0.0.1:80) open
```

```
/ $ nc -vz www.linux.com 80
```

```
nc: bad address 'www.linux.com'
```

```
/ $ exit
```

5. Update the NetworkPolicy and comment out the Egress line. Then replace the policy.

```
student@ckad-1:~/app2$ vim $HOME/app2/allclosed.yaml
```

allclosed.yaml

```yaml
....
spec:
  podSelector: {}
  policyTypes:
  - Ingress
#  - Egress            #<-- Comment out this line
```

```
student@ckad-1:~/app2$ kubectl replace -f $HOME/app2/allclosed.yaml
```

```
networkpolicy.networking.k8s.io/deny-default replaced
```

6. Test egress access to an outside site. Get the IP address of the **eth0** inside the container while logged in. The IP is `192.168.55.91` in the example below, yours may be different.

```
student@ckad-1:~/app2$ kubectl exec -it -c busy secondapp -- sh
```

© Copyright the Linux Foundation 2020. All rights reserved.

On Container

```
/ $ nc -vz www.linux.com 80
```

```
www.linux.com (151.101.185.5:80) open
```

```
/ $ ip a
```

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue 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: tunl0@NONE: <NOARP> mtu 1480 qdisc noop qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
4: eth0@if59: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 1e:c8:7d:6a:96:c3 brd ff:ff:ff:ff:ff:ff
    inet 192.168.55.91/32 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::1cc8:7dff:fe6a:96c3/64 scope link
       valid_lft forever preferred_lft forever
```

```
/ $ exit
```

7. Now add a selector to allow ingress to only the nginx container. Use the IP from the **eth0** range.

```
student@ckad-1:~/app2$ vim $HOME/app2/allclosed.yaml
```

allclosed.yaml

```
<output_omitted>
 policyTypes:
  - Ingress
  ingress:                    #<-- Add this and following three lines
  - from:
    - ipBlock:
        cidr: 192.168.0.0/16
# - Egress
```

8. Recreate the policy, and verify its configuration.

```
student@ckad-1:~/app2$ kubectl replace -f $HOME/app2/allclosed.yaml
```

```
networkpolicy.networking.k8s.io/deny-default replaced
```

```
student@ckad-1:~/app2$ kubectl get networkpolicy
```

```
NAME           POD-SELECTOR     AGE
deny-default   <none>           3m2s
```

```
student@ckad-1:~/app2$ kubectl get networkpolicy -o yaml
```

```
apiVersion: v1
items:
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
  metadata:
<output_omitted>
```

9. Test access to the container both using **curl** as well as **ping**, the IP address to use was found from the IP inside the container. You may need to install **iputils-ping** or other software to use **ping**.

```
student@ckad-1:~/app2$ curl http://192.168.55.91
```

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

```
student@ckad-1:~/app2$ ping -c5 192.168.55.91
```

```
PING 192.168.55.91 (192.168.55.91) 56(84) bytes of data.
64 bytes from 192.168.55.91: icmp_seq=1 ttl=63 time=1.11 ms
64 bytes from 192.168.55.91: icmp_seq=2 ttl=63 time=0.352 ms
64 bytes from 192.168.55.91: icmp_seq=3 ttl=63 time=0.350 ms
64 bytes from 192.168.55.91: icmp_seq=4 ttl=63 time=0.359 ms
64 bytes from 192.168.55.91: icmp_seq=5 ttl=63 time=0.295 ms

--- 192.168.55.91 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4054ms
rtt min/avg/max/mdev = 0.295/0.495/1.119/0.312 ms
```

10. Update the policy to only allow ingress for TCP traffic on port 80, then test with **curl**, which should work. The ports entry should line up with the from entry a few lines above.

```
student@ckad-1:~/app2$ vim $HOME/app2/allclosed.yaml
```

allclosed.yaml

```yaml
<output_omitted>
  - Ingress
  ingress:
  - from:
    - ipBlock:
        cidr: 192.168.0.0/16
    ports:                     #<-- Add this and two following lines
    - port: 80
      protocol: TCP
#  - Egress
```

```
student@ckad-1:~/app2$ kubectl replace -f $HOME/app2/allclosed.yaml
```

```
networkpolicy.networking.k8s.io/deny-default replaced
```

```
student@ckad-1:~/app2$ curl http://192.168.55.91
```

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

11. All five pings should fail, with zero received.

© Copyright the Linux Foundation 2020. All rights reserved.

```
student@ckad-1:~/app2$ ping -c5 192.168.55.91
```

```
PING 192.168.55.91 (192.168.55.91) 56(84) bytes of data.

--- 192.168.55.91 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4098ms
```

12. You may want to remove the `default-deny` policy, in case you want to get to your registry or other pods.

```
student@ckad-1:~/app2$ kubectl delete networkpolicies deny-default
```

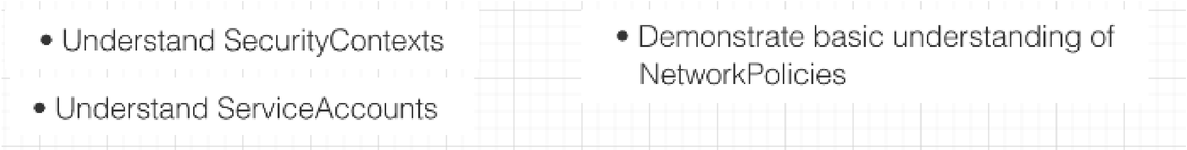
```
networkpolicy.networking.k8s.io "deny-default" deleted
```

Exercise 6.6: Domain Review

Very Important

The source pages and content in this review could change at any time. **IT IS YOUR RESPONSIBILITY TO CHECK THE CURRENT INFORMATION**.

Revisit the CKAD domain list on Curriculum Overview and locate some of the topics we have covered in this chapter.



- Understand SecurityContexts
- Understand ServiceAccounts
- Demonstrate basic understanding of NetworkPolicies

Figure 6.1: **Domain**

Focus on ensuing you have all the necessary files and processes understood first. Repeat the review until you are sure you have bookmarks of necessary YAML samples and can complete each step quickly, and ensure each object is running properly.

1. Create a new deployment which uses the `nginx` image.
2. Create a new `LoadBalancer` service to expose the newly created deployment. Test that it works.
3. Create a new NetworkPolicy called netblock which blocks all traffic to pods in this deployment only. Test that all traffic is blocked to deployment.
4. Update the `netblock` policy to allow traffic to the pod on port 80 only. Test that you can access the default nginx web page.
5. Find and use the `security-review1.yaml` file to create a pod.

```
student@ckad-1:~$ kubectl create -f security-review1.yaml
```

6. View the status of the pod.
7. Use the following commands to figure out why the pod has issues.

```
student@ckad-1:~$ kubectl get pod securityreview

student@ckad-1:~$ kubectl describe pod securityreview

student@ckad-1:~$ kubectl logs securityreview
```

8. After finding the errors, log into the container and find the proper id of the nginx user.

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

9. Edit the yaml and re-create the pod such that the pod runs without error.

10. Create a new `serviceAccount` called securityaccount.

11. Create a ClusterRole named secrole which only allows create, delete, and list of pods in all apiGroups.

12. Bind the clusterRole to the serviceAccount.

13. Locate the token of the securityaccount. Create a file called `/tmp/securitytoken`. Put only the value of `token:` is equal to, a long string that may start with eyJh and be several lines long. Careful that only that string exists in the file.

14. Remove any resources you have added during this review

Ghazela Technology
Academy

Copyright © 2020   adresse : 23, rue el Yakoute ,cité la Ghazelle ,Ariana
MF :1469595/X/A/M/000 – RC : B01912050
Contact : contact at ghazelatc.com
Téléphone :+216 71 866 142 , +216 27 862 155, +216 54 828 018

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

Chapter 7

Exposing Applications

Exercise 7.1: Exposing Applications: Expose a Service

Overview

In this lab we will explore various ways to expose an application to other pods and outside the cluster. We will add to the NodePort used in previous labs other service options.

1. We will begin by using the default service type ClusterIP. This is a cluster internal IP, only reachable from within the cluster. Begin by viewing the existing services.

```
student@ckad-1:~$ kubectl get svc
```

```
NAME          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes    ClusterIP   10.96.0.1        <none>        443/TCP        8d
nginx         ClusterIP   10.108.95.67     <none>        443/TCP        8d
registry      ClusterIP   10.105.119.236   <none>        5000/TCP       8d
secondapp     NodePort    10.111.26.8      <none>        80:32000/TCP   7h
```

2. Delete the existing service for secondapp.

```
student@ckad-1:~$ cd ~/app2
```

```
student@ckad-1:~/app2$ kubectl delete svc secondapp
```

```
service "secondapp" deleted
```

3. Create a YAML file for a replacement service, which would be persistent. Use the label to select the secondapp. Expose the same port and protocol of the previous service.

```
student@ckad-1:~/app2$ vim service.yaml
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: secondapp
  labels:
    run: my-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
  type: NodePort
  selector:
    example: second
```

4. Create the service, find the new IP and port. Note there is no high number port as this is internal access only.

```
student@ckad-1:~/app2$ kubectl create -f service.yaml
```

```
service/secondapp created
```

```
student@ckad-1:~/app2$ kubectl get svc
```

```
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)      AGE
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP      8d
nginx        ClusterIP   10.108.95.67     <none>        443/TCP      8d
registry     ClusterIP   10.105.119.236   <none>        5000/TCP     8d
secondapp    NodePort    10.98.148.52     <none>        80:32212/TCP 14s
```

5. Test access. You should see the default welcome page again.

```
student@ckad-1:~/app2$ curl http://10.98.148.52
```

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

6. To expose a port to outside the cluster we will create a NodePort. We had done this in a previous step from the command line. When we create a NodePort it will create a new ClusterIP automatically. Edit the YAML file again. Add type: NodePort. Also add the high-port to match an open port in the firewall as mentioned in the previous chapter. You'll have to delete and re-create as the existing IP is immutable, but not able to be reused. The NodePort will try to create a new ClusterIP instead.

```
student@ckad-1:~/app2$ vim service.yaml
```

service.yaml

```yaml
....
    protocol: TCP
    nodePort: 32000         #<-- Add this and following line
  type: NodePort
  selector:
    example: second
```

```
student@ckad-1:~/app2$ kubectl delete svc secondapp ; kubectl create  -f service.yaml
```

```
service "secondapp" deleted
service/secondapp created
```

7. Find the new ClusterIP and ports for the service.

```
student@ckad-1:~/app2$ kubectl get svc
```

```
NAME         TYPE         CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP    10.96.0.1      <none>        443/TCP        8d
nginx        ClusterIP    10.108.95.67   <none>        443/TCP        8d
registry     ClusterIP    10.105.119.236 <none>        5000/TCP       8d
secondapp    NodePort     10.109.134.221 <none>        80:32000/TCP   4s
```

8. Test the low port number using the new ClusterIP for the secondapp service.

```
student@ckad-1:~/app2$ curl 10.109.134.221
```

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

9. Test access from an external node to the host IP and the high container port. Your IP and port will be different. It should work, even with the network policy in place, as the traffic is arriving via a 192.168.0.0 port.

```
user@laptop:~/Desktop$ curl http://35.184.219.5:32000
```

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

10. The use of a LoadBalancer makes an asynchronous request to an external provider for a load balancer if one is available. It then creates a NodePort and waits for a response including the external IP. The local NodePort will work even before the load balancer replies. Edit the YAML file and change the type to be LoadBalancer.

```
student@ckad-1:~/app2$ vim service.yaml
```

service.yaml

```
....
  - port: 80
    protocol: TCP
    nodePort: 32000
  type: LoadBalancer     #<-- Edit this line
  selector:
    example: second
```

```
student@ckad-1:~/app2$ kubectl delete svc secondapp ; kubectl create -f service.yaml
```

```
service "secondapp" deleted
service/secondapp created
```

11. As mentioned the cloud provider is not configured to provide a load balancer; the External-IP will remain in pending state. Some issues have been found using this with VirtualBox.

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
student@ckad-1:~/app2$ kubectl get svc
```

```
NAME          TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)         AGE
kubernetes    ClusterIP      10.96.0.1        <none>         443/TCP         8d
nginx         ClusterIP      10.108.95.67     <none>         443/TCP         8d
registry      ClusterIP      10.105.119.236   <none>         5000/TCP        8d
secondapp     LoadBalancer   10.109.26.21     <pending>      80:32000/TCP    4s
```

12. Test again local and from a remote node. The IP addresses and ports will be different on your node.

```
serewic@laptop:~/Desktop$ curl http://35.184.219.5:32000
```

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

13. You can also use DNS names provided by **CoreDNS** which dynamically are added when the service is created. Start by logging into the `busy` container of secondapp.

```
student@ckad-1:~/app2$ kubectl exec -it secondapp -c busy -- sh
```

On Container

(a) Use the **nslookup** command to find the `secondapp` service. Then find the `registry` service we configured to provide container images.

```
/ $ nslookup secondapp
```

```
Server:         10.96.0.10
Address:        10.96.0.10:53

Name:        secondapp.default.svc.cluster.local
Address: 10.96.214.133

*** Can't find secondapp.svc.cluster.local: No answer
*** Can't find secondapp.cluster.local: No answer
*** Can't find secondapp.c.endless-station-188822.internal: No answer
<output_omitted>
```

```
/ $ nslookup registry
```

```
Server:         10.96.0.10
Address:        10.96.0.10:53

Name:        registry.default.svc.cluster.local
Address: 10.110.95.21
<output_omitted>
```

(b) Lookup the FQDN associated with the DNS server IP displayed by the commands. Your IP may be different.

```
/ $ nslookup 10.96.0.10
```

```
Server:         10.96.0.10
Address:        10.96.0.10:53

10.0.96.10.in-addr.arpa       name = kube-dns.kube-system.svc.cluster.local
```

© Copyright the Linux Foundation 2020. All rights reserved.

(c) Attempt to resolve the service name, which should not bring back any records. Then try with the FQDN. Read through the errors. You'll note that only the default namespaces is checked. You may have to check the FQDN a few times as it doesn't always reply with an answer.

```
/ $ nslookup kube-dns
```

```
Server:         10.96.0.10
Address:        10.96.0.10:53

** server can't find kube-dns.default.svc.cluster.local: NXDOMAIN

*** Can't find kube-dns.svc.cluster.local: No answer
*** Can't find kube-dns.cluster.local: No answer
*** Can't find kube-dns.c.endless-station-188822.internal: No answer
```

```
/ $ nslookup kube-dns.kube-system.svc.cluster.local
```

```
Server:         10.96.0.10
Address:        10.96.0.10:53

Name:        kube-dns.kube-system.svc.cluster.local
Address: 10.96.0.10

*** Can't find kube-dns.kube-system.svc.cluster.local: No answer
```

(d) Exit out of the container

```
/ $ exit
```

14. Create a new namespace named `multitenant` and a new deployment named `mainapp`. Expose the deployment port 80 using the name `shopping`

```
student@ckad-1:~/app2$ kubectl create ns multitenant
student@ckad-1:~/app2$ kubectl -n multitenant create deployment mainapp --image=nginx
student@ckad-1:~/app2$ kubectl -n multitenant expose deployment mainapp --name= shopping \
                          --type=NodePort --port=80
```

15. Log back into the secondapp busy container and test access to mainapp.

```
student@ckad-1:~/app2$ exec -it secondapp -c busy -- sh
```

On Container

(a) Use **nslookup** to determine the address of the new service. Start with using just the service name. Then add the service name and the namespaces. As we have not configured coreDNS at all it may not know about all the possible domains to serve.

```
/ $ nslookup shopping
```

```
Server:         10.96.0.10
Address:        10.96.0.10:53

** server can't find shopping.default.svc.cluster.local: NXDOMAIN

*** Can't find shopping.svc.cluster.local: No answer
<output_omitted>
```

```
/ $ nslookup shopping.multitenant
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
Server:         10.96.0.10
Address:        10.96.0.10:53

** server can't find shopping.multitenant: NXDOMAIN

*** Can't find shopping.multitenant: No answer
```

```
/ $ nslookup shopping.multitenant.svc.cluster.local
```

```
Server:         10.96.0.10
Address:        10.96.0.10:53

*** Can't find shopping.multitenant.svc.cluster.local: No answer

*** Can't find shopping.multitenant.svc.cluster.local: No answer
```

(b) Now try to use the service name and then the name with namespace, to see if it works. The DNS using the namespace should work, even if you don't have access to the default page. RBAC could be used to grant access.

```
/ $wget shopping
```

```
wget: bad address 'shopping'
```

```
/ $wget shopping.multitenant
```

```
Connecting to shopping.multitenant (10.101.4.142:80)
wget: can't open 'index.html': Permission denied
```

Exercise 7.2: Ingress Controller

If you have a large number of services to expose outside of the cluster, or to expose a low-number port on the host node you can deploy an ingress controller. While nginx and GCE have controllers officially supported by Kubernetes.io, the Traefik Ingress Controller is easier to install. At the moment.

1. 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@ckad-1:~/app2$ vim ingress.rbac.yaml
```

ingress.rbac.yaml

```yaml
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: traefik-ingress-controller
rules:
  - apiGroups:
      - ""
    resources:
      - services
      - endpoints
      - secrets
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - extensions
    resources:
      - ingresses
    verbs:
      - get
      - list
      - watch
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
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
```

2. Create the new role and binding.

```
student@ckad-1:~/app2$ kubectl create -f ingress.rbac.yaml
```

```
clusterrole.rbac.authorization.k8s.io/traefik-ingress-controller created
clusterrolebinding.rbac.authorization.k8s.io/traefik-ingress-controller created
```

3. Create the Traefik controller. The source web page changes on a regular basis. You can find a recent release by going here https://github.com/containous/traefik/releases, The recent 2.X release has many changes and some ”undocumented features” being worked on. Find a copy of the file in the course tarball using the **find** command.

```
student@ckad-1:~/app2$ find $HOME -name traefik-ds.yaml
```

4. The output below represents the changes in a **diff** output, from a downloaded version to the edited file in the tarball. You do not have to do this step, it is so you understand the kind of changes one may need for the dashboard to work. One line was added, six lines removed. Also with version 2.0 the dashboard does not appear to work, so we are declaring the use of version 1.7.13.

```
student@ckad-1:~/app2$ diff download.yaml traefik-ds.yaml
```

traefik-ds.yaml

```
23a24              ##  Add the following line 24
>       hostNetwork: true
34,39d34         ##  Remove these lines around line 34
<         securityContext:
<           capabilities:
<             drop:
<             - ALL
<             add:
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
<             - NET_BIND_SERVICE
```

The included file looks like this:

traefik-ds.rule.yaml

```
....
      terminationGracePeriodSeconds: 60
      hostNetwork: True
      containers:
      - image: traefik
        name: traefik-ingress-lb
        ports:
        - name: http
          containerPort: 80
          hostPort: 80
        - name: admin
          containerPort: 8080
          hostPort: 8080
        args:
        - --api
....
```

5. Create the objects using the edited file.

```
student@ckad-1:~/app2$ kubectl apply -f traefik-ds.yaml
```

```
serviceaccount/traefik-ingress-controller created
daemonset.apps/traefik-ingress-controller created
service/traefik-ingress-service created
```

6. 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 we've been working with.

```
student@ckad-1:~/app2$ vim ingress.rule.yaml
```

ingress.rule.yaml

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-test
  annotations:
    kubernetes.io/ingress.class: traefik
spec:
  rules:
  - host: www.example.com
    http:
      paths:
      - backend:
          serviceName: secondapp
          servicePort: 80
        path: /
```

© Copyright the Linux Foundation 2020. All rights reserved.

7. Now ingest the rule into the cluster.

```
student@ckad-1:~/app2$ kubectl create -f ingress.rule.yaml
```

```
ingress.networking.k8s.io/ingress-test2 created
```

8. 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 load balancer. In this example the interface would be `ens4`, and the IP would be `10.128.0.7`.

```
student@ckad-1:~$ 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@ckad-1:~/app2$ curl -H "Host: www.example.com" http://10.128.0.7/
```

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

```
user@laptop:~$ curl -H "Host: www.example.com" http://35.193.3.179
```

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

9. 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@ckad-1:~/app2$ kubectl create deployment thirdpage --image=nginx
```

```
deployment.apps "thirdpage" created
```

10. Assign a label for the ingress controller to match against. Your pod name is unique, you can use the **Tab** key to complete the name.

```
student@ckad-1:~/app2$ kubectl label pod thirdpage-<tab> example=third
```

11. Expose the new server as a NodePort.

```
student@ckad-1:~/app2$ kubectl expose deployment thirdpage --port=80 --type=NodePort
```

```
service/thirdpage exposed
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

12. Now we will customize the installation. Run a bash shell inside the new pod. Your pod name will end differently. Install **vim** or an editor inside the container then edit the `index.html` file of nginx so that the title of the web page will be `Third Page`. Much of the command output is not shown below.

```
student@ckad-1:~/app2$ kubectl exec -it thirdpage-<Tab> -- /bin/bash
```

On Container

```
root@thirdpage-:/# apt-get update

root@thirdpage-:/# apt-get install vim -y

root@thirdpage-:/# vim /usr/share/nginx/html/index.html
```

```
<!DOCTYPE html>
<html>
<head>
<title>Third Page</title>     #<-- Edit this line
<style>
<output_omitted>
```

```
root@thirdpage-:/$ exit
```

Edit the ingress rules to point the thirdpage service.

13. `student@ckad-1:~/app2$ kubectl edit ingress ingress-test`

ingress test

```yaml
....
spec:
  rules:
  - host: www.example.com
    http:
      paths:
      - backend:
          serviceName: secondapp
          servicePort: 80
        path: /
        pathType: ImplementationSpecific
  - host: thirdpage.org                   #<<-- Add this and the following 7 lines
    http:
      paths:
      - backend:
          serviceName: thirdpage
          servicePort: 80
        path: /
        pathType: ImplementationSpecific
status:
....
```

14. Test the second `Host:` setting using **curl** locally as well as from a remote system, be sure the `<title>` shows the non-default page. Use the main IP of either node.

```
student@ckad-1:~/app2$ curl -H "Host: thirdpage.org" http://10.128.0.7/
```

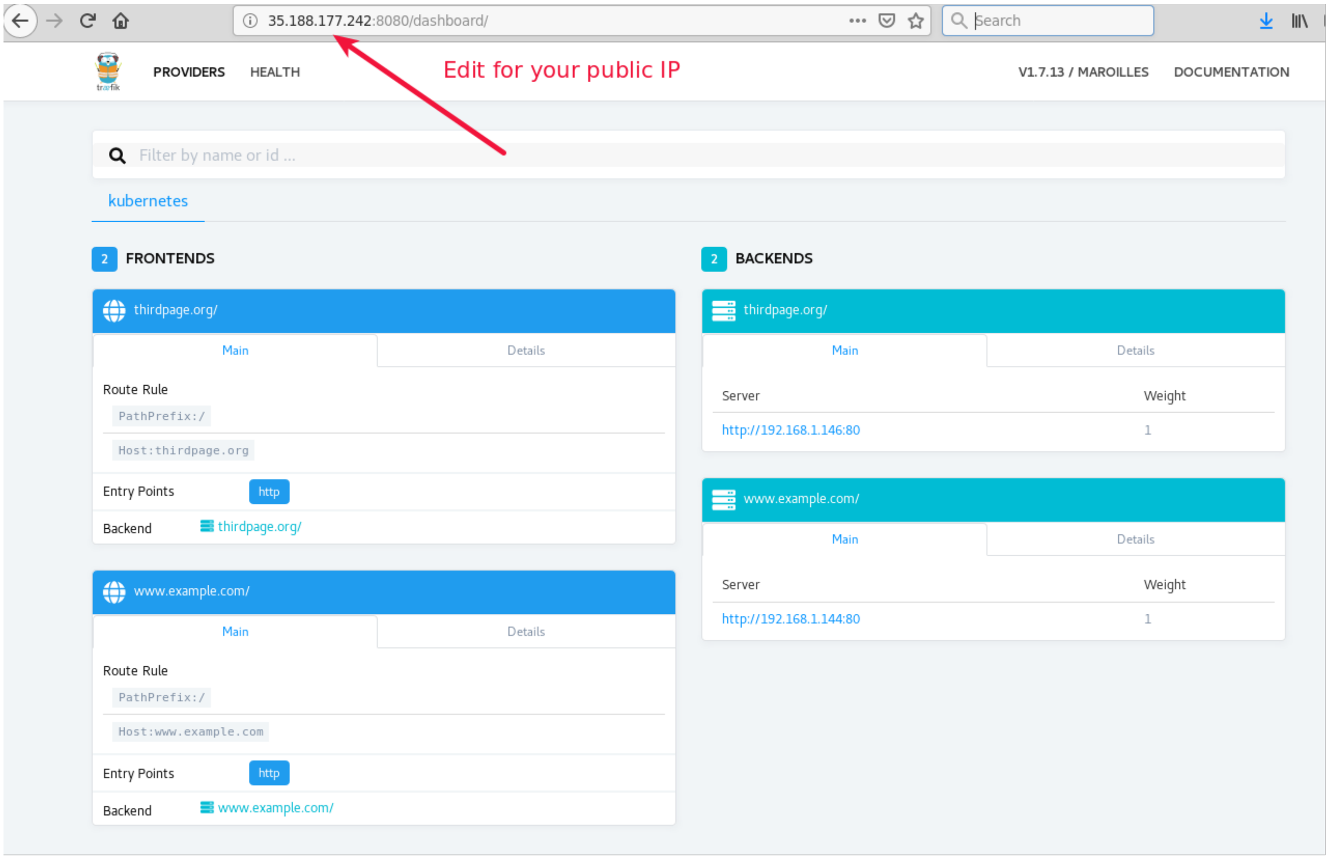
```
<!DOCTYPE html>
<html>
```

© Copyright the Linux Foundation 2020. All rights reserved.

```
<head>
<title>Third Page</title>
<style>
<output_omitted>
```

15. The **Traefik.io** ingress controller also presents a dashboard which allows you to monitor basic traffic. From your local system open a browser and navigate to the public IP of your master node with a like this ⟨YOURPUBLICIP⟩:8080/dashboard/. The trailing slash makes a difference.

    Follow the `HEALTH` and `PROVIDERS` links at the top, as well as the the node IP links and you can view traffic when you reference the pages, from inside or outside the node. Typo the domain names inside the **curl** command and you can also see `404 error traffic`. Explore as time permits.



Edit for your public IP

Figure 7.1: **Accessing the API**

```
student@ckad-1:~/app2$ curl -H "Host: thirdpage.org" http://10.128.0.7/

student@ckad-1:~/app2$ curl -H "Host: nopage.net" http://10.128.0.7/
```

Exercise 7.3: Domain Review

Very Important

The source pages and content in this review could change at any time. **IT IS YOUR RESPONSIBILITY TO CHECK THE CURRENT INFORMATION.**

Revisit the CKAD domain list on Curriculum Overview and locate some of the topics we have covered in this chapter.

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

- Understand Services

Figure 7.2: **Service Domain Topic**

Focus on ensuing you have all the necessary files and processes understood first. Repeat the review until you are sure you have bookmarks of necessary YAML samples and can complete each step quickly, and ensure each object is running properly.

Using the browser and the three URL locations allowed by the exam, find and bookmark working YAML examples to do the following:

1. Create a new pod called `webone`, running the `nginx` service. Expose port 80.
2. Create a new service named `webone-svc`. The service should be accessible from outside the cluster.
3. Update both the pod and the service with selectors so that traffic for to the service IP shows the web server content.
4. Change the type of the service such that it is only accessible from within the cluster. Test that exterior access no longer works, but access from within the node works.
5. Deploy another pod, called `webtwo`, this time running the `wlniao/website` image. Create another service, called `webtwo-svc` such that only requests from within the cluster work. Note the default page for each server is distinct.
6. Install and configure an ingress controller such that requests for webone.com see the `nginx` default page, and requests for webtwo.org see the `wlniao/website` default page.
7. Remove any resources created in this review.

Ghazela Technology
Academy

Copyright © 2020 adresse : 23, rue el Yakoute ,cité la Ghazelle ,Ariana
MF :1469595/X/A/M/000 – RC : B01912050
Contact : contact at ghazelatc.com
Téléphone :+216 71 866 142 , +216 27 862 155, +216 54 828 018

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

Chapter 8

Troubleshooting

Exercise 8.1: Troubleshooting: Monitor Applications

Overview

Troubleshooting can be difficult in a multi-node, decoupled and transient environment. Add in the rapid pace of change and it becomes more difficult. Instead of focusing and remembering a particular error and the fix it may be more useful to learn a flow of troubleshooting and revisit assumptions until the pace of change slows and various areas further mature.

1. View the `secondapp` pod, it should show as `Running`. This may not mean the application within is working properly, but that the pod is running. The restarts are due to the command we have written to run. The pod exists when done, and the controller restarts another container inside. The count depends on how long the labs have been running.

```
student@ckad-1/app2:~$ cd
student@ckad-1:~$ kubectl get pods secondapp
```

```
NAME                          READY     STATUS      RESTARTS    AGE
secondapp                     2/2       Running     49          2d
```

2. Look closer at the pod. Working slowly through the output check each line. If you have issues, are other pods having issues on the same node or volume? Check the state of each container. Both `busy` and `webserver` should report as `Running`. Note `webserver` has a restart count of zero while `busy` has a restart count of 49. We expect this as, in our case, the pod has been running for 49 hours.

```
student@ckad-1:~$ kubectl describe pod secondapp
```

```
Name:         secondapp
Namespace:    default
Node:         ckad-2-wdrq/10.128.0.2
Start Time:   Fri, 13 Apr 2018 20:34:56 +0000
Labels:       example=second
Annotations:  <none>
Status:       Running
IP:           192.168.55.91
Containers:
  webserver:
```

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
<output_omitted>
    State:          Running
      Started:      Fri, 13 Apr 2018 20:34:58 +0000
    Ready:          True
    Restart Count:  0
<output_omitted>

  busy:
<output_omitted>

    State:          Running
      Started:      Sun, 15 Apr 2018 21:36:20 +0000
    Last State:     Terminated
      Reason:       Completed
      Exit Code:    0
      Started:      Sun, 15 Apr 2018 20:36:18 +0000
      Finished:     Sun, 15 Apr 2018 21:36:18 +0000
    Ready:          True
    Restart Count:  49
    Environment:    <none>
```

3. There are three values for conditions. Check that the pod reports Initialized, Ready and scheduled.

```
<output_omitted>
Conditions:
  Type           Status
  Initialized    True
  Ready          True
  PodScheduled   True
<output_omitted>
```

4. Check if there are any events with errors or warnings which may indicate what is causing any problems.

```
Events:
  Type    Reason   Age                From                  Message
  ----    ------   ----               ----                  -------
  Normal  Pulling  34m (x50 over 2d)  kubelet, ckad-2-wdrq  pulling
image "busybox"
  Normal  Pulled   34m (x50 over 2d)  kubelet, ckad-2-wdrq  Successfully
pulled image "busybox"
  Normal  Created  34m (x50 over 2d)  kubelet, ckad-2-wdrq  Created
container
  Normal  Started  34m (x50 over 2d)  kubelet, ckad-2-wdrq  Started
container
```

5. View each container log. You may have to sift errors from expected output. Some containers may have no output at all, as is found with busy.

```
student@ckad-1:~$ kubectl logs secondapp webserver
```

```
192.168.55.0 - - [13/Apr/2018:21:18:13 +0000] "GET / HTTP/1.1" 200
612 "-" "curl/7.47.0" "-"
192.168.55.0 - - [13/Apr/2018:21:20:35 +0000] "GET / HTTP/1.1" 200
612 "-" "curl/7.53.1" "-"
127.0.0.1 - - [13/Apr/2018:21:25:29 +0000] "GET" 400 174 "-" "-" "-"
127.0.0.1 - - [13/Apr/2018:21:26:19 +0000] "GET index.html" 400 174
"-" "-" "-"
<output_omitted>
```

```
student@ckad-1:~$ kubectl logs secondapp busy
```

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
student@ckad-1:˜$
```

6. Check to make sure the container is able to use DNS and communicate with the outside world. Remember we still have limited the UID for `secondapp` to be UID **2000**, which may prevent some commands from running. It can also prevent an application from completing expected tasks, and other errors.

```
student@ckad-1:˜$ kubectl exec  -it  secondapp -c busy -- sh
```

On Container

```
/ $ nslookup www.linuxfoundation.org
```

```
/ $ nslookup www.linuxfoundation.org
Server:                 10.96.0.10
Address:        10.96.0.10:53

Non-authoritative answer:
Name:        www.linuxfoundation.org
Address: 23.185.0.2

*** Can't find www.linuxfoundation.org: No answer
```

```
/ $ cat /etc/resolv.conf
```

```
nameserver 10.96.0.10
search default.svc.cluster.local svc.cluster.local
cluster.local c.endless-station-188822.internal
google.internal
options ndots:5
```

Test access to a remote node using **nc (NetCat)**. There are several options to **nc** which can help troubleshoot if the problem is the local node, something between nodes or in the target. In the example below the connect never completes and a **control-c** was used to interrupt.

7. `/ $ nc www.linux.com 25`

```
^Cpunt!
```

8. Test using an IP address in order to narrow the issue to name resolution. In this case the IP in use is a well known IP for Google's DNS servers. The following example shows that Internet name resolution is working, but our UID issue prevents access to the `index.html` file.

```
/ $ wget http://www.linux.com/
```

```
Connecting to www.linux.com (151.101.45.5:80)
Connecting to www.linux.com (151.101.45.5:443)
wget: can't open 'index.html': Permission denied

/ $ exit
```

9. Make sure traffic is being sent to the correct Pod. Check the details of both the service and endpoint. Pay close attention to ports in use as a simple typo can prevent traffic from reaching the proper pod. Make sure labels and selectors don't have any typos as well.

```
student@ckad-1:˜$ kubectl get svc
```

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
NAME         TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP      10.96.0.1        <none>        443/TCP        10d
nginx        ClusterIP      10.108.95.67     <none>        443/TCP        10d
registry     ClusterIP      10.105.119.236   <none>        5000/TCP       10d
secondapp    LoadBalancer   10.109.26.21     <pending>     80:32000/TCP   1d
thirdpage    NodePort       10.109.250.78    <none>        80:31230/TCP   1h
```

```
student@ckad-1:~$ kubectl get svc secondapp -o yaml
```

```
<output_omitted>
  clusterIP: 10.109.26.21
  externalTrafficPolicy: Cluster
  ports:
  - nodePort: 32000
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    example: second
<output_omitted>
```

10. Verify an endpoint for the service exists and has expected values, including namespaces, ports and protocols.

```
student@ckad-1:~$ kubectl get ep
```

```
NAME         ENDPOINTS            AGE
kubernetes   10.128.0.3:6443      10d
nginx        192.168.55.68:443    10d
registry     192.168.55.69:5000   10d
secondapp    192.168.55.91:80     1d
thirdpage    192.168.241.57:80    1h
```

```
student@ckad-1:~$ kubectl get ep secondapp -o yaml
```

```
apiVersion: v1
kind: Endpoints
metadata:
  creationTimestamp: 2018-04-14T05:37:32Z
<output_omitted>
```

11. If the containers, services and endpoints are working the issue may be with an infrastructure service like **kube-proxy**. Ensure it's running, then look for errors in the logs. As we have two nodes we will have two proxies to look at. As we built our cluster with **kubeadm** the proxy runs as a container. On other systems you may need to use **journalctl** or look under `/var/log/kube-proxy.log`.

```
student@ckad-1:~$ ps -elf |grep kube-proxy
```

```
4 S root      2864  2847  0  80   0 - 14178 -      15:45 ?
 00:00:56 /usr/local/bin/kube-proxy --config=/var/lib/kube-proxy/config.conf
0 S student  23513 18282  0  80   0 -  3236 pipe_w 22:49 pts/0
 00:00:00 grep --color=auto kube-proxy
```

```
student@ckad-1:~$ journalctl -a | grep proxy
```

```
Apr 15 15:44:43 ckad-2-nzjr audit[742]: AVC apparmor="STATUS"
 operation="profile_load" profile="unconfined" \
  name="/usr/lib/lxd/lxd-bridge-proxy" pid=742 comm="apparmor_parser"
Apr 15 15:44:43 ckad-2-nzjr kernel: audit: type=1400
 audit(1523807083.011:11): apparmor="STATUS" \
  operation="profile_load" profile="unconfined" \
```

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
   name="/usr/lib/lxd/lxd-bridge-proxy" pid=742 comm="apparmor_parser"
Apr 15 15:45:17 ckad-2-nzjr kubelet[1248]: I0415 15:45:17.153670
 1248 reconciler.go:217] operationExecutor.VerifyControllerAttachedVolume\
  started for volume "xtables-lock" \
   (UniqueName: "kubernetes.io/host-path/e701fc01-38f3-11e8-a142-\
   42010a800003-xtables-lock") \
   pod "kube-proxy-t8k4w" (UID: "e701fc01-38f3-11e8-a142-42010a800003")
```

12. Look at both of the proxy logs. Lines which begin with the character **I** are `info`, **E** are `errors`. In this example the last message says access to listing an endpoint was denied by RBAC. It was because a default installation via Helm wasn't RBAC aware. If not using command line completion, view the possible pod names first.

```
student@ckad-1:~$ kubectl -n kube-system get pod
```

```
student@ckad-1:~$ kubectl -n kube-system logs kube-proxy-fsdfr
```

```
I0405 17:28:37.091224       1 feature_gate.go:190] feature gates: map[]
W0405 17:28:37.100565       1 server_others.go:289] Flag proxy-mode=""
unknown, assuming iptables proxy
I0405 17:28:37.101846       1 server_others.go:138] Using iptables Proxier.
I0405 17:28:37.121601       1 server_others.go:171] Tearing down
inactive rules.
<output_omitted>
E0415 15:45:17.086081       1 reflector.go:205] \
  k8s.io/kubernetes/pkg/client/informers/informers_generated/
  internalversion/factory.go:85: \
  Failed to list *core.Endpoints: endpoints is forbidden: \
    User "system:serviceaccount:kube-system:kube-proxy" cannot \
   list endpoints at the cluster scope:\
 [clusterrole.rbac.authorization.k8s.io "system:node-proxier" not found, \
   clusterrole.rbac.authorization.k8s.io "system:basic-user" not found,
 clusterrole.rbac.authorization.k8s.io \
 "system:discovery" not found]
```

13. Check that the proxy is creating the expected rules for the problem service. Find the destination port being used for the service, **32000** in this case.

```
student@ckad-1:~$ sudo iptables-save |grep secondapp
```

```
-A KUBE-NODEPORTS -p tcp -m comment --comment "default/secondapp:" \
-m tcp --dport 32000 -j KUBE-MARK-MASQ
-A KUBE-NODEPORTS -p tcp -m comment --comment "default/secondapp:" \
-m tcp --dport 32000 -j KUBE-SVC-DAASHM5XQZF5XI3E
-A KUBE-SERVICES ! -s 192.168.0.0/16 -d 10.109.26.21/32 -p tcp \
-m comment --comment "default/secondapp: \
      cluster IP" -m tcp --dport 80 -j KUBE-MARK-MASQ
-A KUBE-SERVICES -d 10.109.26.21/32 -p tcp -m comment --comment \
"default/secondapp: cluster IP" -m tcp \
      --dport 80 -j KUBE-SVC-DAASHM5XQZF5XI3E
<output_omitted>
```

14. Ensure the proxy is working by checking the port targeted by **iptables**. If it fails open a second terminal and view the proxy logs when making a request as it happens.

```
student@ckad-1:~$ curl localhost:32000
```

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

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

Exercise 8.2: OPTIONAL LAB: Conformance Testing

The **cncf.io** group is in the process of formalizing what is considered to be a conforming Kubernetes cluster. While that project matures there is an existing tool provided by **Heptio** which can be useful. We will need to make sure a newer version of **Golang** is installed for it to work. You can download the code from github and look around with git or with go, depending on which tool you are most familiar. **Things change quickly these steps may not work....today**

1. Download a compiled binary. A shorter URL is shown first, then the longer, just in case the link changes and you need to navigate. They should download the same file.

```
student@ckad-1:~$ curl -sLO https://tinyurl.com/yyu5bs28
```

```
student@ckad-1:~$ mv yyu5bs28 sonobuoy.tar.gz
```

```
student@ckad-1:~$ tar -xvf sonobuoy.tar.gz
```

```
LICENSE
sonobuoy
```

```
student@ckad-1:~$ curl -sLO \
https://github.com/heptio/sonobuoy/releases/download/v0.15.4/sonobuoy_0.15.4_linux_amd64.tar.gz
```

2. Run the test. We will not use the `--wait` option, which will capture the screen until the test finishes. This could take a while to finish. You should get some output indicating testing objects being created.

```
student@ckad-1:~$ sudo mv sonobuoy /usr/local/bin/
```

```
student@ckad-1:~$ sonobuoy run
```

```
WARN[0000] The maximum supported Kubernetes version is 1.15.99, but
the server version is v1.16.1. Sonobuoy will continue but unexpected results may occur.
INFO[0000] created object                           name=sonobuoy namespace= resource=namespaces
INFO[0000] created object                           name=sonobuoy-serviceaccount namespace=sonobuoy ....
INFO[0000] created object                           name=sonobuoy-serviceaccount-sonobuoy namespace=....
INFO[0000] created object                           name=sonobuoy-serviceaccount namespace= resource....
INFO[0000] created object                           name=sonobuoy-config-cm namespace=sonobuoy resou....
INFO[0000] created object                           name=sonobuoy-plugins-cm namespace=sonobuoy reso....
INFO[0000] created object                           name=sonobuoy namespace=sonobuoy resource=pods
INFO[0000] created object                           name=sonobuoy-master namespace=sonobuoy resource....
```

3. View the results inside the `sonobuoy` pod.

```
student@ckad-1:~$ kubectl get pods --all-namespaces
```

```
<output_omitted>
sonobuoy      sonobuoy                                                 1/1
     Running   0          90s
sonobuoy      sonobuoy-e2e-job-b3bcb52b4fd54367                        2/2
     Running   0          85s
sonobuoy      sonobuoy-systemd-logs-daemon-set-f7ca2bb9a7174908-h47kb   2/2     Running   0          85s
sonobuoy      sonobuoy-systemd-logs-daemon-set-f7ca2bb9a7174908-s22d6   2/2     Running   0          85s
```

```
student@ckad-1:~$ kubectl -n sonobuoy exec -it sonobuoy -- /bin/bash
```

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

On Container

4. View the files inside the container.

```
root@sonobuoy:/# ls
```

```
bin   home   mnt         root                        sbin      tmp
boot  lib    opt         run                         sonobuoy  usr
dev   lib64  plugins.d   run_master.sh               srv       var
etc   media  proc        run_single_node_worker.sh   sys
```

5. View the `run_master.sh` script. Note that it mentions both the **sonobuoy** command and where to find the results.

```
root@sonobuoy:/# cat run_master.sh
```

```
#!/bin/bash
##########################################################################
# Copyright 2017 Heptio Inc.
#

<output_omitted>
RESULTS_DIR="${RESULTS_DIR:-/tmp/sonobuoy}"
# It's ok for these env vars to be unbound
RESULTS_DIR="${RESULTS_DIR}" SONOBUOY_CONFIG="${SONOBUOY_CONFIG}"
SONOBUOY_ADVERTISE_IP="${SONOBUOY_ADVERTISE_IP}" /sonobuoy master -v 3 --logtostderr

echo -n "${RESULTS_DIR}/$(ls -t "${RESULTS_DIR}" | grep -v done | head -n 1)" > "${RESULTS_DIR}"/done
```

6. View the contents of the `/tmp/sonobuoy` directory. Note the subdirectory is a generated number, yours will be different. The **Tab** key can be used to complete the path.

```
root@sonobuoy:/# ls /tmp/sonobuoy/
```

```
d39f2629-fa3c-4a0b-9b33-53080e78b57b
```

```
root@sonobuoy:/# cd /tmp/sonobuoy/d39f2629-fa3c-4a0b-9b33-53080e78b57b ; ls
```

```
meta  plugins
```

```
root@sonobuoy:...57b# find .
```

```
.
./plugins
./plugins/systemd-logs
./plugins/systemd-logs/results
./plugins/systemd-logs/results/e-6clr
./plugins/systemd-logs/results/e-6clr/systemd_logs
./plugins/systemd-logs/results/e-5c7t
./plugins/systemd-logs/results/e-5c7t/systemd_logs
./meta
./meta/run.log
./meta/config.json
```

7. The **sonobuoy** command has several options. We will use two to explore the test output.

```
root@sonobuoy:...57b# cd /
```

```
root@sonobuoy:/# ./sonobuoy status
```

© Copyright the Linux Foundation 2020. All rights reserved.

© The Linux Foundation 2020: DO NOT COPY OR DISTRIBUTE

```
        PLUGIN     STATUS   RESULT   COUNT
           e2e    running                1
  systemd-logs   complete                2

Sonobuoy is still running. Runs can take up to 60 minutes.
```

```
root@sonobuoy:/# ./sonobuoy logs
```

```
<output_omitted>
```

8. Continue to look through tests and results as time permits. Connect to the other pods in the `sonobuoy` namespace and look for log and result files.

   There is also an online, graphical scanner. In testing, inside GCE, the results were blocked and never returned. You may have different outcome in other environments.

Exercise 8.3: Domain Review

Very Important

The source pages and content in this review could change at any time. **IT IS YOUR RESPONSIBILITY TO CHECK THE CURRENT INFORMATION.**

Revisit the CKAD domain list on Curriculum Overview and locate some of the topics we have covered in this chapter.

- Understand debugging in Kubernetes

Figure 8.1: **Troubleshooting Domain Topic**

Focus on ensuing you have all the necessary files and processes understood first. Repeat the review until you are sure you have bookmarks of necessary YAML samples and can complete each step quickly, and ensure each object is running properly.

1. Find and use the `troubleshoot-review1.yaml` file to create a deployment. The **create** command will fail. Edit the file to fix issues such that a single pod runs for at least a minute without issue. There are several things to fix.

```
student@ckad-1:˜$ kubectl create -f troubleshoot-review1.yaml
```

```
<Fix any errors found here>
```

   When fixed it should look like this:

```
student@ckad-1:˜$ kubectl get deploy igottrouble
```

```
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
igottrouble   1/1     1            1           5m13s
```

2. Remove any resources created during this review.

© Copyright the Linux Foundation 2020. All rights reserved.