Linux Academy CKA Notes :

## Upgrading the Kubernetes Cluster :

----------------------------------------------------------------------------

Kubeadm allows us to upgrade our cluster components in the proper order, making sure to include important feature upgrades we might want to take advantage of in the latest stable version of Kubernertes. In this lesson, we will go through upgrading our cluster from version 1.13.5 to 1.14.1.

Get the version of the API server:

kubectl version --short

View the version of kubelet:

kubectl describe nodes

View the version of controller-manager pod:

kubectl get po [controller\_pod\_name] -o yaml -n kube-system

Release the hold on versions of kubeadm and kubelet:

sudo apt-mark unhold kubeadm kubelet

Install version 1.14.1 of kubeadm:

sudo apt install -y kubeadm=1.14.1-00

Hold the version of kubeadm at 1.14.1:

sudo apt-mark hold kubeadm

Verify the version of kubeadm:

kubeadm version

Plan the upgrade of all the controller components:

sudo kubeadm upgrade plan

Upgrade the controller components:

sudo kubeadm upgrade apply v1.14.1

Release the hold on the version of kubectl:

sudo apt-mark unhold kubectl

Upgrade kubectl:

sudo apt install -y kubectl=1.14.1-00

Hold the version of kubectl at 1.14.1:

sudo apt-mark hold kubectl

Upgrade the version of kubelet:

sudo apt install -y kubelet=1.14.1-00

Hold the version of kubelet at 1.14.1:

sudo apt-mark hold kubelet

**Helpful Links**

* [Upgrading Kubernetes](https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm-upgrade/)
* [Changelog for v1.13](https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG-1.13.md)

Simple Steps :

On master:

Unhold kubeadm,kubelet version

Upgrade kubeadm on Master node , hold the version upgrade the control plain components.

Unhold the kubectl version , Upgrade the kubectl , hold the new version again , upgrade the kubelet hold the version.

On Workers:

Unhold kubeadm,kubelet version

Upgrade kubeadm on worker node , hold the version

Unhold the kubectl version , Upgrade the kubectl , hold the new version again , upgrade the kubelet hold the version.

**Operating System Upgrades within a Kubernetes Cluster**

When we need to take a node down for maintenance, Kubernetes makes it easy to evict the pods on that node, take it down, and then continue scheduling pods after the maintenance is complete. Furthermore, if the node needs to be decommissioned, you can just as easily remove the node and replace it with a new one, joining it to the cluster.

See which pods are running on which nodes:

kubectl get pods -o wide

Evict the pods on a node:

kubectl drain [node\_name] --ignore-daemonsets

Watch as the node changes status:

kubectl get nodes -w

Schedule pods to the node after maintenance is complete:

kubectl uncordon [node\_name]

Remove a node from the cluster:

kubectl delete node [node\_name]

Generate a new token:

sudo kubeadm token generate

List the tokens:

sudo kubeadm token list

Print the kubeadm join command to join a node to the cluster:

sudo kubeadm token create [token\_name] --ttl 2h --print-join-command

**Helpful Links**

* [Maintenance on a Node](https://kubernetes.io/docs/tasks/administer-cluster/cluster-management/#maintenance-on-a-node)

## Backing Up and Restoring a Kubernetes Cluster :

Backing up your cluster can be a useful exercise, especially if you have a single etcd cluster, as all the cluster state is stored there. The etcdctl utility allows us to easily create a snapshot of our cluster state (etcd) and save this to an external location. In this lesson, we’ll go through creating the snapshot and talk about restoring in the event of failure.

Get the etcd binaries:

wget https://github.com/etcd-io/etcd/releases/download/v3.3.12/etcd-v3.3.12-linux-amd64.tar.gz

Unzip the compressed binaries:

tar xvf etcd-v3.3.12-linux-amd64.tar.gz

Move the files into /usr/local/bin:

sudo mv etcd-v3.3.12-linux-amd64/etcd\* /usr/local/bin

Take a snapshot of the etcd datastore using etcdctl:

sudo ETCDCTL\_API=3 etcdctl snapshot save snapshot.db --cacert /etc/kubernetes/pki/etcd/server.crt --cert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/ca.key

View the help page for etcdctl:

ETCDCTL\_API=3 etcdctl --help

Browse to the folder that contains the certificate files:

cd /etc/kubernetes/pki/etcd/

View that the snapshot was successful:

ETCDCTL\_API=3 etcdctl --write-out=table snapshot status snapshot.db

Zip up the contents of the etcd directory:

sudo tar -zcvf etcd.tar.gz /etc/kubernetes/pki/etcd

Copy the etcd directory to another server:

scp etcd.tar.gz cloud\_user@18.219.235.42:~/

**Helpful Links**

* [Backing up the etcd Store](https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/#backing-up-an-etcd-cluster)
* [etcd Disaster Recovery Examples](https://github.com/etcd-io/etcd/blob/master/Documentation/op-guide/recovery.md)

## Pod and Node Networking

Kubernetes keeps networking simple for effective communication between pods, even if they are located on a different node. In this lesson, we’ll talk about pod communication from within a node, including how to inspect the virtual interfaces, and then get into what happens when a pod wants to talk to another pod on a different node.

See which node our pod is on:

kubectl get pods -o wide

Log in to the node:

ssh [node\_name]

View the node's virtual network interfaces:

ifconfig

View the containers in the pod:

docker ps

Get the process ID for the container:

docker inspect --format '{{ .State.Pid }}' [container\_id]

Use nsenter to run a command in the process's network namespace:

nsenter -t [container\_pid] -n ip addr

**Helpful Links**

* [Cluster Networking](https://kubernetes.io/docs/concepts/cluster-administration/networking/)

## Container Network Interface (CNI)

A Container Network Interface (CNI) is an easy way to ease communication between containers in a cluster. The CNI has many responsibilities, including IP management, encapsulating packets, and mappings in userspace. In this lesson, we will cover the details of the Flannel CNI we used in our Linux Academy cluster and talk about the ways in which we simplified communication in our cluster.

Apply the Flannel CNI plugin:

kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/bc79dd1505b0c8681ece4de4c0d86c5cd2643275/Documentation/kube-flannel.yml

**Helpful Links**

* [Flannel Documentation](https://github.com/coreos/flannel/blob/master/Documentation/kubernetes.md)
* [Installing Other CNI Plugins](https://kubernetes.io/docs/setup/independent/create-cluster-kubeadm/#pod-network)
* [Installing Addons in Kubernetes](https://kubernetes.io/docs/concepts/cluster-administration/addons/)

## Service Networking

Services allow our pods to move around, get deleted, and replicate, all without having to manually keep track of their IP addresses in the cluster. This is accomplished by creating one gateway to distribute packets evenly across all pods. In this lesson, we will see the differences between a NodePort service and a ClusterIP service and see how the iptables rules take effect when traffic is coming in.

YAML for the nginx NodePort service:

apiVersion: v1

kind: Service

metadata:

name: nginx-nodeport

spec:

type: NodePort

ports:

- protocol: TCP

port: 80

targetPort: 80

nodePort: 30080

selector:

app: nginx

Get the services YAML output for all the services in your cluster:

kubectl get services -o yaml

Try and ping the clusterIP service IP address:

ping 10.96.0.1

View the list of services in your cluster:

kubectl get services

View the list of endpoints in your cluster that get created with a service:

kubectl get endpoints

Look at the iptables rules for your services:

sudo iptables-save | grep KUBE | grep nginx

**Helpful Links**

* [Services in Kubernetes](https://kubernetes.io/docs/concepts/services-networking/service/)

## Ingress Rules and Load Balancers

When handling traffic from outside sources, there are two ways to direct that traffic to your pods: deploying a load balancer, and creating an ingress controller and an Ingress resource. In this lesson, we will talk about the benefits of each and how Kubernetes distributes traffic to the pods on a node to reduce latency and direct traffic to the appropriate services within your cluster.

View the list of services:

kubectl get services

The load balancer YAML spec:

apiVersion: v1

kind: Service

metadata:

name: nginx-loadbalancer

spec:

type: LoadBalancer

ports:

- port: 80

targetPort: 80

selector:

app: nginx

Create a new deployment:

kubectl run kubeserve2 --image=chadmcrowell/kubeserve2

View the list of deployments:

kubectl get deployments

Scale the deployments to 2 replicas:

kubectl scale deployment/kubeserve2 --replicas=2

View which pods are on which nodes:

kubectl get pods -o wide

Create a load balancer from a deployment:

kubectl expose deployment kubeserve2 --port 80 --target-port 8080 --type LoadBalancer

View the services in your cluster:

kubectl get services

Watch as an external port is created for a service:

kubectl get services -w

Look at the YAML for a service:

kubectl get services kubeserve2 -o yaml

Curl the external IP of the load balancer:

curl http://[external-ip]

View the annotation associated with a service:

kubectl describe services kubeserve

Set the annotation to route load balancer traffic local to the node:

kubectl annotate service kubeserve2 externalTrafficPolicy=Local

The YAML for an Ingress resource:

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: service-ingress

spec:

rules:

- host: kubeserve.example.com

http:

paths:

- backend:

serviceName: kubeserve2

servicePort: 80

- host: app.example.com

http:

paths:

- backend:

serviceName: nginx

servicePort: 80

- http:

paths:

- backend:

serviceName: httpd

servicePort: 80

Edit the ingress rules:

kubectl edit ingress

View the existing ingress rules:

kubectl describe ingress

Curl the hostname of your Ingress resource:

curl http://kubeserve2.example.com

**Helpful Links**

* [Create an External Load Balancer](https://kubernetes.io/docs/tasks/access-application-cluster/create-external-load-balancer/)
* [Ingress](https://kubernetes.io/docs/concepts/services-networking/ingress/)

## Cluster DNS

CoreDNS is now the new default DNS plugin for Kubernetes. In this lesson, we’ll go over the hostnames for pods and services. We will also discover how you can customize DNS to include your own nameservers.

View the CoreDNS pods in the kube-system namespace:

kubectl get pods -n kube-system

View the CoreDNS deployment in your Kubernetes cluster:

kubectl get deployments -n kube-system

View the service that performs load balancing for the DNS server:

kubectl get services -n kube-system

Spec for the busybox pod:

apiVersion: v1

kind: Pod

metadata:

name: busybox

namespace: default

spec:

containers:

- image: busybox:1.28.4

command:

- sleep

- "3600"

imagePullPolicy: IfNotPresent

name: busybox

restartPolicy: Always

View the resolv.conf file that contains the nameserver and search in DNS:

kubectl exec -it busybox -- cat /etc/resolv.conf

Look up the DNS name for the native Kubernetes service:

kubectl exec -it busybox -- nslookup kubernetes

Look up the DNS names of your pods:

kubectl exec -ti busybox -- nslookup [pod-ip-address].default.pod.cluster.local

Look up a service in your Kubernetes cluster:

kubectl exec -it busybox -- nslookup kube-dns.kube-system.svc.cluster.local

Get the logs of your CoreDNS pods:

kubectl logs [coredns-pod-name]

YAML spec for a headless service:

apiVersion: v1

kind: Service

metadata:

name: kube-headless

spec:

clusterIP: None

ports:

- port: 80

targetPort: 8080

selector:

app: kubserve2

YAML spec for a custom DNS pod:

apiVersion: v1

kind: Pod

metadata:

namespace: default

name: dns-example

spec:

containers:

- name: test

image: nginx

dnsPolicy: "None"

dnsConfig:

nameservers:

- 8.8.8.8

searches:

- ns1.svc.cluster.local

- my.dns.search.suffix

options:

- name: ndots

value: "2"

- name: edns0

**Helpful Links**

* [DNS for Services and Pods](https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/)
* [Debugging DNS Resolution](https://kubernetes.io/docs/tasks/administer-cluster/dns-debugging-resolution/)
* [Customizing DNS](https://kubernetes.io/docs/tasks/administer-cluster/dns-custom-nameservers/)
* [CoreDNS GitHub](https://github.com/coredns/deployment/tree/master/kubernetes)
* [Kubernetes DNS-Based Service Discovery](https://github.com/kubernetes/dns/blob/master/docs/specification.md)
* [Deploying CoreDNS using kubeadm](https://coredns.io/2018/01/29/deploying-kubernetes-with-coredns-using-kubeadm/)

## Configuring the Kubernetes Scheduler

The default scheduler in Kubernetes attempts to find the best node for your pod by going through a series of steps. In this lesson, we will cover the steps in detail in order to better understand the scheduler’s function when placing pods on nodes to maximize uptime for the applications running in your cluster. We will also go through how to create a deployment with node affinity.

Label your node as being located in availability zone 1:

kubectl label node chadcrowell1c.mylabserver.com availability-zone=zone1

Label your node as dedicated infrastructure:

kubectl label node chadcrowell2c.mylabserver.com share-type=dedicated

Here is the YAML for the deployment to include the node affinity rules:

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: pref

spec:

replicas: 5

template:

metadata:

labels:

app: pref

spec:

affinity:

nodeAffinity:

preferredDuringSchedulingIgnoredDuringExecution:

- weight: 80

preference:

matchExpressions:

- key: availability-zone

operator: In

values:

- zone1

- weight: 20

preference:

matchExpressions:

- key: share-type

operator: In

values:

- dedicated

containers:

- args:

- sleep

- "99999"

image: busybox

name: main

Create the deployment:

kubectl create -f pref-deployment.yaml

View the deployment:

kubectl get deployments

View which pods landed on which nodes:

kubectl get pods -o wide

#### Helpful Links

* [Assigning a Pod to a Node](https://kubernetes.io/docs/concepts/configuration/assign-pod-node/)
* [Pod and Node Affinity Rules](https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity)

**Running Multiple Schedulers for Multiple Pods**

In Kubernetes, you can run multiple schedulers simultaneously. You can then use different schedulers to schedule different pods. You may, for example, want to set different rules for the scheduler to run all of your pods on one node. In this lesson, I will show you how to deploy a new scheduler alongside your default scheduler and then schedule three different pods using the two schedulers.

**ClusterRole.yaml**

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: ClusterRole

metadata:

name: csinodes-admin

rules:

- apiGroups: ["storage.k8s.io"]

resources: ["csinodes"]

verbs: ["get", "watch", "list"]

**ClusterRoleBinding.yaml**

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: read-csinodes-global

subjects:

- kind: ServiceAccount

name: my-scheduler

namespace: kube-system

roleRef:

kind: ClusterRole

name: csinodes-admin

apiGroup: rbac.authorization.k8s.io

**Role.yaml**

apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

name: system:serviceaccount:kube-system:my-scheduler

namespace: kube-system

rules:

- apiGroups:

- storage.k8s.io

resources:

- csinodes

verbs:

- get

- list

- watch

**RoleBinding.yaml**

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: read-csinodes

namespace: kube-system

subjects:

- kind: User

name: kubernetes-admin

apiGroup: rbac.authorization.k8s.io

roleRef:

kind: Role

name: system:serviceaccount:kube-system:my-scheduler

apiGroup: rbac.authorization.k8s.io

**Edit the existing kube-scheduler cluster role with kubectl edit clusterrole system:kube-scheduler and add the following:**

- apiGroups:

- ""

resourceNames:

- kube-scheduler

- my-scheduler

resources:

- endpoints

verbs:

- delete

- get

- patch

- update

- apiGroups:

- storage.k8s.io

resources:

- storageclasses

verbs:

- watch

- list

- get

**My-scheduler.yaml**

apiVersion: v1

kind: ServiceAccount

metadata:

name: my-scheduler

namespace: kube-system

---

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: my-scheduler-as-kube-scheduler

subjects:

- kind: ServiceAccount

name: my-scheduler

namespace: kube-system

roleRef:

kind: ClusterRole

name: system:kube-scheduler

apiGroup: rbac.authorization.k8s.io

---

apiVersion: apps/v1

kind: Deployment

metadata:

labels:

component: scheduler

tier: control-plane

name: my-scheduler

namespace: kube-system

spec:

selector:

matchLabels:

component: scheduler

tier: control-plane

replicas: 1

template:

metadata:

labels:

component: scheduler

tier: control-plane

version: second

spec:

serviceAccountName: my-scheduler

containers:

- command:

- /usr/local/bin/kube-scheduler

- --address=0.0.0.0

- --leader-elect=false

- --scheduler-name=my-scheduler

image: chadmcrowell/custom-scheduler

livenessProbe:

httpGet:

path: /healthz

port: 10251

initialDelaySeconds: 15

name: kube-second-scheduler

readinessProbe:

httpGet:

path: /healthz

port: 10251

resources:

requests:

cpu: '0.1'

securityContext:

privileged: false

volumeMounts: []

hostNetwork: false

hostPID: false

volumes: []

**Run the deployment for my-scheduler:**

kubectl create -f my-scheduler.yaml

**View your new scheduler in the kube-system namespace:**

kubectl get pods -n kube-system

**pod1.yaml**

apiVersion: v1

kind: Pod

metadata:

name: no-annotation

labels:

name: multischeduler-example

spec:

containers:

- name: pod-with-no-annotation-container

image: k8s.gcr.io/pause:2.0

**pod2.yaml**

apiVersion: v1

kind: Pod

metadata:

name: annotation-default-scheduler

labels:

name: multischeduler-example

spec:

schedulerName: default-scheduler

containers:

- name: pod-with-default-annotation-container

image: k8s.gcr.io/pause:2.0

**pod3.yaml**

apiVersion: v1

kind: Pod

metadata:

name: annotation-second-scheduler

labels:

name: multischeduler-example

spec:

schedulerName: my-scheduler

containers:

- name: pod-with-second-annotation-container

image: k8s.gcr.io/pause:2.0

View the pods as they are created:

kubectl get pods -o wide

**Helpful Links**

* [Configure Multiple Schedulers](https://kubernetes.io/docs/tasks/administer-cluster/configure-multiple-schedulers/)

## Scheduling Pods with Resource Limits and Label Selectors

In order to share the resources of your node properly, you can set resource limits and requests in Kubernetes. This allows you to reserve enough CPU and memory for your application while still maintaining system health. In this lesson, we will create some requests and limits in our pod YAML to show how it’s used by the node.

View the capacity and the allocatable info from a node:

kubectl describe nodes

The pod YAML for a pod with requests:

apiVersion: v1

kind: Pod

metadata:

name: resource-pod1

spec:

nodeSelector:

kubernetes.io/hostname: "chadcrowell3c.mylabserver.com"

containers:

- image: busybox

command: ["dd", "if=/dev/zero", "of=/dev/null"]

name: pod1

resources:

requests:

cpu: 800m

memory: 20Mi

Create the requests pod:

kubectl create -f resource-pod1.yaml

View the pods and nodes they landed on:

kubectl get pods -o wide

The YAML for a pod that has a large request:

apiVersion: v1

kind: Pod

metadata:

name: resource-pod2

spec:

nodeSelector:

kubernetes.io/hostname: "chadcrowell3c.mylabserver.com"

containers:

- image: busybox

command: ["dd", "if=/dev/zero", "of=/dev/null"]

name: pod2

resources:

requests:

cpu: 1000m

memory: 20Mi

Create the pod with 1000 millicore request:

kubectl create -f resource-pod2.yaml

See why the pod with a large request didn’t get scheduled:

kubectl describe resource-pod2

Look at the total requests per node:

kubectl describe nodes chadcrowell3c.mylabserver.com

Delete the first pod to make room for the pod with a large request:

kubectl delete pods resource-pod1

Watch as the first pod is terminated and the second pod is started:

kubectl get pods -o wide -w

The YAML for a pod that has limits:

apiVersion: v1

kind: Pod

metadata:

name: limited-pod

spec:

containers:

- image: busybox

command: ["dd", "if=/dev/zero", "of=/dev/null"]

name: main

resources:

limits:

cpu: 1

memory: 20Mi

Create a pod with limits:

kubectl create -f limited-pod.yaml

Use the exec utility to use the top command:

kubectl exec -it limited-pod top

#### Helpful Links

* [Configure Default CPU Requests and Limits](https://kubernetes.io/docs/tasks/administer-cluster/manage-resources/cpu-default-namespace/)
* [Configure Default Memory Requests and Limits](https://kubernetes.io/docs/tasks/administer-cluster/manage-resources/memory-default-namespace/)

## DaemonSets and Manually Scheduled Pods

DaemonSets do not use a scheduler to deploy pods. In fact, there are currently DaemonSets in the Kubernetes cluster that we made. In this lesson, I will show you where to find those and how to create your own DaemonSet pods to deploy without the need for a scheduler.

Find the DaemonSet pods that exist in your kubeadm cluster:

kubectl get pods -n kube-system -o wide

Delete a DaemonSet pod and see what happens:

kubectl delete pods [pod\_name] -n kube-system

Give the node a label to signify it has SSD:

kubectl label node[node\_name] disk=ssd

The YAML for a DaemonSet:

apiVersion: apps/v1beta2

kind: DaemonSet

metadata:

name: ssd-monitor

spec:

selector:

matchLabels:

app: ssd-monitor

template:

metadata:

labels:

app: ssd-monitor

spec:

nodeSelector:

disk: ssd

containers:

- name: main

image: linuxacademycontent/ssd-monitor

Create a DaemonSet from a YAML spec:

kubectl create -f ssd-monitor.yaml

Label another node to specify it has SSD:

kubectl label node chadcrowell2c.mylabserver.com disk=ssd

View the DaemonSet pods that have been deployed:

kubectl get pods -o wide

Remove the label from a node and watch the DaemonSet pod terminate:

kubectl label node chadcrowell3c.mylabserver.com disk-

Change the label on a node to change it to spinning disk:

kubectl label node chadcrowell2c.mylabserver.com disk=hdd --overwrite

Pick the label to choose for your DaemonSet:

kubectl get nodes chadcrowell3c.mylabserver.com --show-labels

**Helpful Links**

* [DaemonSets](https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/)

**Displaying Scheduler Events**

There are multiple ways to view the events related to the scheduler. In this lesson, we’ll look at ways in which you can troubleshoot any problems with your scheduler or just find out more information.

View the name of the scheduler pod:

kubectl get pods -n kube-system

Get the information about your scheduler pod events:

kubectl describe pods [scheduler\_pod\_name] -n kube-system

View the events in your default namespace:

kubectl get events

View the events in your kube-system namespace:

kubectl get events -n kube-system

Delete all the pods in your default namespace:

kubectl delete pods --all

Watch events as they are appearing in real time:

kubectl get events -w

View the logs from the scheduler pod:

kubectl logs [kube\_scheduler\_pod\_name] -n kube-system

The location of a systemd service scheduler pod:

/var/log/kube-scheduler.log

**Helpful Links**

* [Verify the Desired Scheduler](https://kubernetes.io/docs/tasks/administer-cluster/configure-multiple-schedulers/#verifying-that-the-pods-were-scheduled-using-the-desired-schedulers)

## Deploying an Application, Rolling Updates, and Rollbacks

We already know Kubernetes will run pods and deployments, but what happens when you need to update or change the version of your application running inside of the Kubernetes cluster? That’s where rolling updates come in, allowing you to update the app image with zero downtime. In this lesson, we’ll go over a rolling update, how to roll back, and how to pause the update if things aren’t going well.

The YAML for a deployment:

apiVersion: apps/v1

kind: Deployment

metadata:

name: kubeserve

spec:

replicas: 3

selector:

matchLabels:

app: kubeserve

template:

metadata:

name: kubeserve

labels:

app: kubeserve

spec:

containers:

- image: linuxacademycontent/kubeserve:v1

name: app

Create a deployment with a record (for rollbacks):

kubectl create -f kubeserve-deployment.yaml --record

Check the status of the rollout:

kubectl rollout status deployments kubeserve

View the ReplicaSets in your cluster:

kubectl get replicasets

Scale up your deployment by adding more replicas:

kubectl scale deployment kubeserve --replicas=5

Expose the deployment and provide it a service:

kubectl expose deployment kubeserve --port 80 --target-port 80 --type NodePort

Set the minReadySeconds attribute to your deployment:

kubectl patch deployment kubeserve -p '{"spec": {"minReadySeconds": 10}}'

Use kubectl apply to update a deployment:

kubectl apply -f kubeserve-deployment.yaml

Use kubectl replace to replace an existing deployment:

kubectl replace -f kubeserve-deployment.yaml

Run this curl look while the update happens:

while true; do curl http://10.105.31.119; done

Perform the rolling update:

kubectl set image deployments/kubeserve app=linuxacademycontent/kubeserve:v2 --v 6

Describe a certain ReplicaSet:

kubectl describe replicasets kubeserve-[hash]

Apply the rolling update to version 3 (buggy):

kubectl set image deployment kubeserve app=linuxacademycontent/kubeserve:v3

Undo the rollout and roll back to the previous version:

kubectl rollout undo deployments kubeserve

Look at the rollout history:

kubectl rollout history deployment kubeserve

Roll back to a certain revision:

kubectl rollout undo deployment kubeserve --to-revision=2

Pause the rollout in the middle of a rolling update (canary release):

kubectl rollout pause deployment kubeserve

Resume the rollout after the rolling update looks good:

kubectl rollout resume deployment kubeserve

### Helpful Links

* [Deployments](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/)
* [Creating a Deployment](https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/)
* [Performing a Rolling Update](https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/)

## Configuring an Application for High Availability and Scale

Continuing from the last lesson, we will go through how Kubernetes will save you from EVER releasing code with bugs. Then, we will talk about ConfigMaps and secrets as a way to pass configuration data to your apps.

The YAML for a readiness probe:

apiVersion: apps/v1

kind: Deployment

metadata:

name: kubeserve

spec:

replicas: 3

selector:

matchLabels:

app: kubeserve

minReadySeconds: 10

strategy:

rollingUpdate:

maxSurge: 1

maxUnavailable: 0

type: RollingUpdate

template:

metadata:

name: kubeserve

labels:

app: kubeserve

spec:

containers:

- image: linuxacademycontent/kubeserve:v3

name: app

readinessProbe:

periodSeconds: 1

httpGet:

path: /

port: 80

Apply the readiness probe:

kubectl apply -f kubeserve-deployment-readiness.yaml

View the rollout status:

kubectl rollout status deployment kubeserve

Describe deployment:

kubectl describe deployment

Create a ConfigMap with two keys:

kubectl create configmap appconfig --from-literal=key1=value1 --from-literal=key2=value2

Get the YAML back out from the ConfigMap:

kubectl get configmap appconfig -o yaml

The YAML for the ConfigMap pod:

apiVersion: v1

kind: Pod

metadata:

name: configmap-pod

spec:

containers:

- name: app-container

image: busybox:1.28

command: ['sh', '-c', "echo $(MY\_VAR) && sleep 3600"]

env:

- name: MY\_VAR

valueFrom:

configMapKeyRef:

name: appconfig

key: key1

Create the pod that is passing the ConfigMap data:

kubectl apply -f configmap-pod.yaml

Get the logs from the pod displaying the value:

kubectl logs configmap-pod

The YAML for a pod that has a ConfigMap volume attached:

apiVersion: v1

kind: Pod

metadata:

name: configmap-volume-pod

spec:

containers:

- name: app-container

image: busybox

command: ['sh', '-c', "echo $(MY\_VAR) && sleep 3600"]

volumeMounts:

- name: configmapvolume

mountPath: /etc/config

volumes:

- name: configmapvolume

configMap:

name: appconfig

Create the ConfigMap volume pod:

kubectl apply -f configmap-volume-pod.yaml

Get the keys from the volume on the container:

kubectl exec configmap-volume-pod -- ls /etc/config

Get the values from the volume on the pod:

kubectl exec configmap-volume-pod -- cat /etc/config/key1

The YAML for a secret:

apiVersion: v1

kind: Secret

metadata:

name: appsecret

stringData:

cert: value

key: value

Create the secret:

kubectl apply -f appsecret.yaml

The YAML for a pod that will use the secret:

apiVersion: v1

kind: Pod

metadata:

name: secret-pod

spec:

containers:

- name: app-container

image: busybox

command: ['sh', '-c', "echo Hello, Kubernetes! && sleep 3600"]

env:

- name: MY\_CERT

valueFrom:

secretKeyRef:

name: appsecret

key: cert

Create the pod that has attached secret data:

kubectl apply -f secret-pod.yaml

Open a shell and echo the environment variable:

kubectl exec -it secret-pod -- sh

echo $MY\_CERT

The YAML for a pod that will access the secret from a volume:

apiVersion: v1

kind: Pod

metadata:

name: secret-volume-pod

spec:

containers:

- name: app-container

image: busybox

command: ['sh', '-c', "echo $(MY\_VAR) && sleep 3600"]

volumeMounts:

- name: secretvolume

mountPath: /etc/certs

volumes:

- name: secretvolume

secret:

secretName: appsecret

Create the pod with volume attached with secrets:

kubectl apply -f secret-volume-pod.yaml

Get the keys from the volume mounted to the container with the secrets:

kubectl exec secret-volume-pod -- ls /etc/certs

#### Helpful Links

* [Scaling Your Application](https://kubernetes.io/docs/concepts/cluster-administration/manage-deployment/#scaling-your-application)
* [Configure Pod ConfigMaps](https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/)
* [Secrets](https://kubernetes.io/docs/concepts/configuration/secret/)

## Creating a Self-Healing Application

In this lesson, we’ll go through the power of ReplicaSets, which make your application self-healing by replicating pods and moving them around and spinning them up when nodes fail. We’ll also talk about StatefulSets and the benefit they provide.

The YAML for a ReplicaSet:

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: myreplicaset

labels:

app: app

tier: frontend

spec:

replicas: 3

selector:

matchLabels:

tier: frontend

template:

metadata:

labels:

tier: frontend

spec:

containers:

- name: main

image: linuxacademycontent/kubeserve

Create the ReplicaSet:

kubectl apply -f replicaset.yaml

The YAML for a pod with the same label as a ReplicaSet:

apiVersion: v1

kind: Pod

metadata:

name: pod1

labels:

tier: frontend

spec:

containers:

- name: main

image: linuxacademycontent/kubeserve

Create the pod with the same label:

kubectl apply -f pod-replica.yaml

Watch the pod get terminated:

kubectl get pods -w

The YAML for a StatefulSet:

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: web

spec:

serviceName: "nginx"

replicas: 2

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

name: web

volumeMounts:

- name: www

mountPath: /usr/share/nginx/html

volumeClaimTemplates:

- metadata:

name: www

spec:

accessModes: [ "ReadWriteOnce" ]

resources:

requests:

storage: 1Gi

Create the StatefulSet:

kubectl apply -f statefulset.yaml

View all StatefulSets in the cluster:

kubectl get statefulsets

Describe the StatefulSets:

kubectl describe statefulsets

#### Helpful Links

* [ReplicaSet](https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/)
* [StatefulSets](https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/)

**Persistent Volumes**

In Kubernetes, pods are ephemeral. This creates a unique challenge with attaching storage directly to the filesystem of a container. Persistent Volumes are used to create an abstraction layer between the application and the underlying storage, making it easier for the storage to follow the pods as they are deleted, moved, and created within your Kubernetes cluster.

In the Google Cloud Engine, find the region your cluster is in:

gcloud container clusters list

Using Google Cloud, create a persistent disk in the same region as your cluster:

gcloud compute disks create --size=1GiB --zone=us-central1-a mongodb

The YAML for a pod that will use persistent disk:

apiVersion: v1

kind: Pod

metadata:

name: mongodb

spec:

volumes:

- name: mongodb-data

gcePersistentDisk:

pdName: mongodb

fsType: ext4

containers:

- image: mongo

name: mongodb

volumeMounts:

- name: mongodb-data

mountPath: /data/db

ports:

- containerPort: 27017

protocol: TCP

Create the pod with disk attached and mounted:

kubectl apply -f mongodb-pod.yaml

See which node the pod landed on:

kubectl get pods -o wide

Connect to the mongodb shell:

kubectl exec -it mongodb mongo

Switch to the mystore database in the mongodb shell:

use mystore

Create a JSON document to insert into the database:

db.foo.insert({name:'foo'})

View the document you just created:

db.foo.find()

Exit from the mongodb shell:

exit

Delete the pod:

kubectl delete pod mongodb

Create a new pod with the same attached disk:

kubectl apply -f mongodb-pod.yaml

Check to see which node the pod landed on:

kubectl get pods -o wide

Drain the node (if the pod is on the same node as before):

kubectl drain [node\_name] --ignore-daemonsets

Once the pod is on a different node, access the mongodb shell again:

kubectl exec -it mongodb mongo

Access the mystore database again:

use mystore

Find the document you created from before:

db.foo.find()

The YAML for a PersistentVolume object in Kubernetes:

apiVersion: v1

kind: PersistentVolume

metadata:

name: mongodb-pv

spec:

capacity:

storage: 1Gi

accessModes:

- ReadWriteOnce

- ReadOnlyMany

persistentVolumeReclaimPolicy: Retain

gcePersistentDisk:

pdName: mongodb

fsType: ext4

Create the Persistent Volume resource:

kubectl apply -f mongodb-persistentvolume.yaml

View our Persistent Volumes:

kubectl get persistentvolumes

**Helpful Links:**

* [Persistent Volumes](https://kubernetes.io/docs/concepts/storage/persistent-volumes/)
* [Configure Persistent Volume Storage](https://kubernetes.io/docs/tasks/configure-pod-container/configure-persistent-volume-storage/)