SEPTEMBER 6, 2021

DevOps (K) Classroom Series – 05/Sept/2021

Kubernetes Installation

* There are many different ways of installing kuberentes and also kuberentes is readily available as a Service on Cloud (Azure Kuberentes Service, Elastic Kuberentes Service, Google Kubernetes Engine)
* To start with we would be using kubeadm as an approach to create a kubernetes cluster
* [Refer Here](https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/) for the installation instructions
* Let’s create 3 ubuntu servers
* Now install docker on the three servers and then execute the commands as mentioned in the document [Refer Here](https://kubernetes.io/docs/setup/production-environment/container-runtimes/#docker)
* Now install Kube-adm, kubectl and kubelet [Refer Here](https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/#installing-kubeadm-kubelet-and-kubectl)
* Now we can create cluster using kubeadm [Refer Here](https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/)
* Login into the master node

sudo -i

kubeadm init --pod-network-cidr=10.244.0.0/16

* Make a note of the instructions given which appear as shown

Your Kubernetes control-plane has initialized successfully!

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

mkdir -p $HOME/.kube

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

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

Alternatively, if you are the root user, you can run:

export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.

Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:

https://kubernetes.io/docs/concepts/cluster-administration/addons/

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

kubeadm join 172.31.37.90:6443 --token q331we.xj6o2j07sy066v1r \

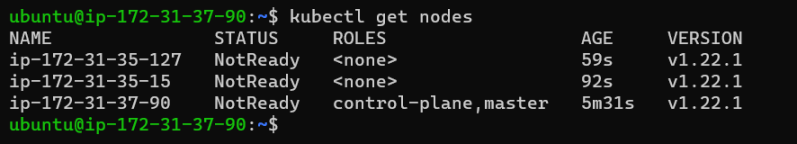
--discovery-token-ca-cert-hash sha256:e281cbdb4c7de0c621e78d42d88b2bfb7d64d454afc56d3090dca3966be76ae3

* Now exit from root user to become normal user and execute the following commands on master

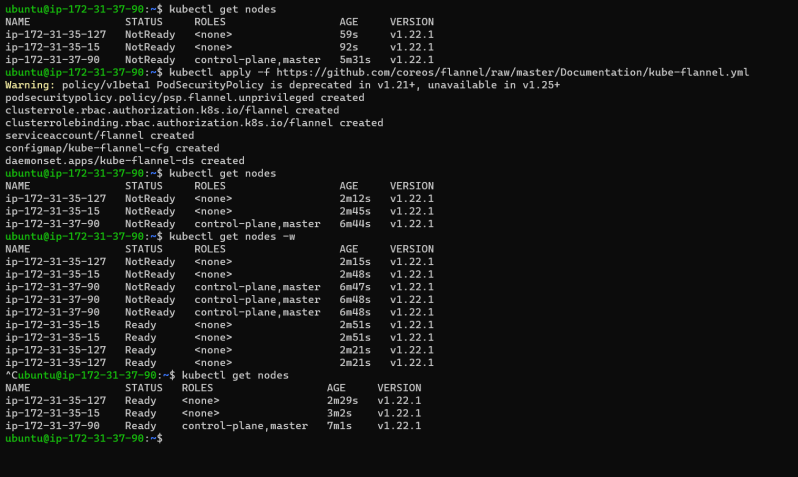
mkdir -p $HOME/.kube

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

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

* Now login into the nodes and execute the kubeadm command as a root user
* Now login into kuberentes (k8s) and execute kubectl get nodes 
* Now execute the following command to install pod network plugin

kubectl apply -f https://github.com/coreos/flannel/raw/master/Documentation/kube-flannel.yml



**Share this:**

SEPTEMBER 7, 2021

DevOps Classroom Series – 07/Sept/2021

Questions Regarding Taking Containers to Production

* We know that we can run our application inside a container, But if we want to go to live/production does containers offer me
  + Fault Tolerance: What happens when the host running the container goes down?
  + High Availability and Scaling: How to handle load during peak time for the applications
  + Manual Interventions: Do i need to scale the applications manually?
  + New Updates: Every application will have a new releases with DevOps quite frequently, how do i update containers to the new versions without having downtime
  + Monitoring capabilities
* We need some kind of orchestration software around containers that will help us solve the above questions.
* That is what Kubernetes does.
* Kubernetes (k8s) takes care of
  + Scaling requirements
  + failover
  + deployment options for zero downtime deployment
* K8s provides
  + Service discovery and load balancing
  + Storage Orchestration
  + Automated rollouts and rollbacks
  + Self-Healing
  + Secret and configuration Management
* k8s is a platform that manages container-based application’s, their networking and storage components
* k8s allows us to focus on application workloads rather than underlying infra
* k8s provides a declarative approach to create deployments
* For more info [Refer Here](https://directdevops.blog/2019/10/09/kubernetes-introduction/)
* For understanding master and node components [Refer Here](https://directdevops.blog/2019/10/10/kubernetes-master-and-node-components/)
* Interactions b/w k8s components

Lab Setup

* We have already installed kubernetes using kubeadm
* Now we if we want to work with kubernetes we will be using kubectl and [Refer Here](https://kubernetes.io/docs/reference/kubectl/cheatsheet/) for its cheat sheet
* Let’s setup auto completion on kubectl

echo "source <(kubectl completion bash)" >> ~/.bashrc

**Share this:**

SEPTEMBER 8, 2021

# DevOps Classroom Series – 08/Sept/2021

## Namespaces

* K8s uses namespaces to organize objects in the cluster.
* Namespace can be thought as a virtual cluster.
* By default kubectl command line interacts with default namespace
* If you want to get the objects from all namespaces

kubectl get <object> --all-namespaces

* If you want to get the object from a specific namespace other than default

kubectl get <object> --namespace <name>

## K8s objects

* k8s objects are persistent entities in k8s system.
* k8s uses these objects to represent state of the cluster
  + what containerized applications are running
  + Resources available to those applications
* To work with k8s objects whether to create, modify, delete, you will need to use the Kubernetes API
* Kubernetes API is HTTP API that API-Server exposes and lets end users or different parts of k8s cluster communicate with one another
* For us we will be using Kubernetes API through the kubectl command-line
* Object Spec and Status
  + Every k8s object includes two nested object fields
    - object spec
    - object status

## Pods

* Pod is the smallest deployable unit in k8s.
* A Pod is group of one or more containers with shared storage and network resources and a specification for how to run containers
* Each container in the Pod runs in its own cgroup, but they share a number of Linux Namespaces
* Applications running in the Pod share same IP address, port space (network namespace) , have the same hostname.
* However, containers running in different Pods are isolated from each other
* Let’s try to write our first Pod nginx pod
* To write the Pod we need to create a manifest yaml file
* Any manifest in k8s will have the following sections

apiVersion:

kind:

metadata:

spec:

* To understand API version we need to know about API groups and versioning
  + API Groups make it easier to extend k8s API. API group is specified in a REST path and in the apiVersion field of the object
  + There are several api groups in k8s
    - The core group: This is legacy group.
      * if the object belongs to core group and has version v1 apiVersion: v1
    - The named groups
      * if the object belongs to apps group and has version v1 apiVersion: apps/v1
  + K8s will have multiple api versions such v1, v1beta1, v1alpha1
* To write the Pod manifest [Refer Here](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.19/#pod-v1-core)

---

apiVersion: v1

kind: Pod

metadata:

name: my-first-pod

spec:

containers:

- name: nginx-container

- image: nginx:1.21.1

Create a file called as nginx-pod.yaml

* Execute the following commands

kubectl apply -f nginx-pod.yaml

kubectl get pods

kubectl get pods -o wide

kubectl describe pods my-first-pod

kubectl delete -f nginx-pod.yaml

#or

kubectl delete pods my-first-pod

* Now let’s look some common commands
  + Viewing logs: kubectl logs <pod-name>
  + Executing commands inside a container with in a pod kubectl exec <pod-name> -- <command>
  + To login into the container in a terminal kubectl exec -it <pod-name> -- bash
* Let’s write one-more manifest for Jenkins

---

apiVersion: v1

kind: Pod

metadata:

name: my-jenkins-pod

spec:

containers:

- name: jenkins

image: jenkins/jenkins:jdk11

* Exercise Work : Create one pod which runs alpine image

---

apiVersion: v1

kind: Pod

metadata:

name: my-first-pod

spec:

containers:

- name: alpine -container

image: alpine:3.12.8

alpine:3.12.8 [**3.12.8**](https://hub.docker.com/layers/alpine/library/alpine/3.12.8/images/sha256-16aea3fc0648559c47f974d7f31096e71b9ec7013224bb29e35cf3db375af976?context=explore)

SEPTEMBER 9, 2021

# DevOps Classroom Series – 09/Sept/2021

## K8s Pod contd

* [Refer Here](https://github.com/asquarezone/DockerZone/commit/481ad926ddd18379bdfee66e06c2761290ca51e9) for the nginx pod added
* Now create a pod

kubectl apply -f nginx-pod.yaml

* Describe and list your pods

kubectl get pods

kubectl describe pods my-first-pod

* Using Port-Forwarding:
  + Later, in this series we will learn how to expose a service to the world to access your application (right approach)
  + Now let’s try use port-forwarding to access the nginx container running inside a pod
* kubectl port-forward my-first-pod 8080:80
  + The above command will forward the port on the loopback address and can be accessed from within the system using “[http://localhost:8080](http://localhost:8080/)`
  + If we need to access the application with port forwarding from external systems
* kubectl port-forward --address 0.0.0.0 my-first-pod 8080:80
  + With this we can access the application by using the http://<publicip&gt;:8080
* [Refer Here](https://github.com/asquarezone/DockerZone/commit/9c65d537b23740f19f880080ae8109c36039ec3d) for the Jenkins pod manifest
* Now let’s add an alpine pod [Refer Here](https://github.com/asquarezone/DockerZone/commit/0b834943ed25b7749c2f9edbe807b6257e76277b)

## Health Checks

* When you run your application as a container in k8’s it is automatically kept alive for you using a process health check. The health check simply ensures that the main process of your application is always running, If it isn’t K8s restarts it.
* However in most cases a simple process check is insufficient, For example if your process has deadlocked and is unable to server request, process health check will believe that your application is healthy since the process is still running
* To address this k8s has introduced health checks for application liveness. Liveness health checks run application-specific logic to verify that application is not just still running but working properly
* [Refer Here](https://github.com/asquarezone/DockerZone/commit/eb968d5efb4a1e467fa7df580b08b1db1904939a) for the nginx pod with liveness probe
* Readiness Probe: K8s makes distinction b/w liveness and readiness. Liveness determines if the application is running properly. Containers that fail liveness checks are restarted.
* Readiness describes when the container is ready to serve user requests. Containers that fail readiness checks are removed from service load balancers.
* Combining the readiness and liveness probes help ensure only healthy containers are running with in the cluster.

---

|  |
| --- |
|  |
| --- |
|  |  | apiVersion: v1 |
|  |  | kind: Pod |
|  |  | metadata: |
|  |  | name: my-jenkins-pod |
|  |  | spec: |
|  |  | containers: |
|  |  | - name: jenkins |
|  |  | image: jenkins/jenkins:jdk11 |
|  |  | ports: |
|  |  | - name: jenkinsport |
|  |  | containerPort: 8080 |
|  |  | protocol: TCP |
|  |  |  |
|  |  |  |

SEPTEMBER 10, 2021

# DevOps Classroom Series – 10/Sept/2021

## Resource Management

* [Refer Here](https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/) for the official docs
* Resource Requests: Minimum Resources
* Resource Limits: Capping Resource Usage
* [Refer Here](https://github.com/asquarezone/DockerZone/commit/dc6a5fff9368b5091e6876dbd9999129ee5e56b1) for the change set containing k8s pod with resource limits

## Labels

* Labels provide identifying metadata for objects.
* These are fundamental quantities of the k8s object that will be used in grouping viewing and operating
* Label is key value pair

app: nginx

appVersion: 1.2.1

qt.com/app-version: 1.2.1

* Lets create a pod

kubectl run nginx --image=nginx:1.21.1 --labels "ver=1,app=nginx"

kubectl run nginx2 --image=nginx:1.21.1 --labels "ver=2,app=nginx"

* Let’s get the pods kubectl get pods --show-labels 
* Label selectors

kubectl get pods --selector="app in (nginx, httpd)"

kubectl get pods --selector="ver in (nginx, httpd)"

kubectl get pods --selector="ver in (1,5)"

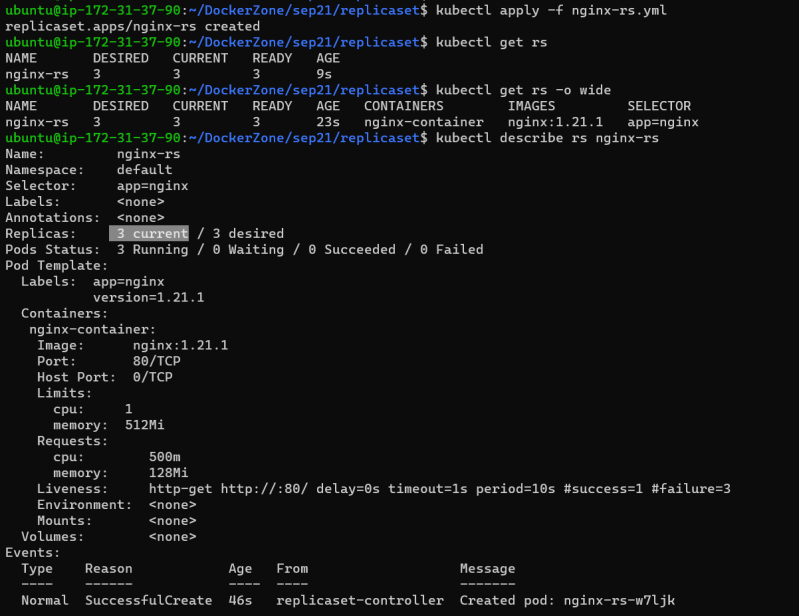
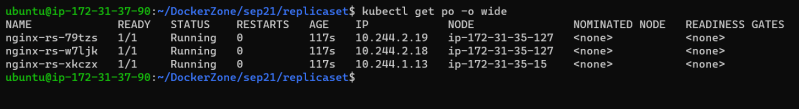
kubectl get pods --selector='app!=nginx'

kubectl get pods --selector="ver notin (1,5)"

kubectl get pods --selector='!env' # doesnt have label with key env

kubectl get pods --selector='env' # has a label env with any value

## Replica Set

* Till now we have run individual container as Pods.
* More often we want multiple replicas of our application to be running
  + Redundancy: mean failure can be tolerated
  + Scale: More requests can be served by your application.
* In Replica Sets, we can specify the desired state in terms of how many replicas we want and kubernetes will ensure that all the replicas are working
* [Refer Here](https://github.com/asquarezone/DockerZone/commit/c71f7ab6420c09b1892ffaa55d00a30eb0e76d17) for the replica set sample and [Refer Here](https://github.com/asquarezone/DockerZone/commit/a91fe9634f0eaa66d0396a6857d69b873be47d9c) for the fix with selector  

### Share this:

SEPTEMBER 13, 2021

# DevOps Classroom Series -12/Sept/2021

## Annotations

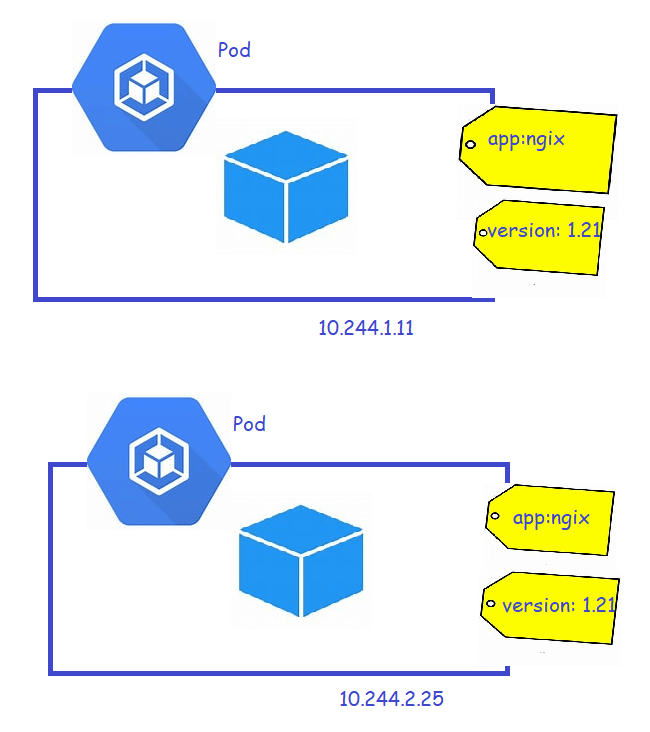
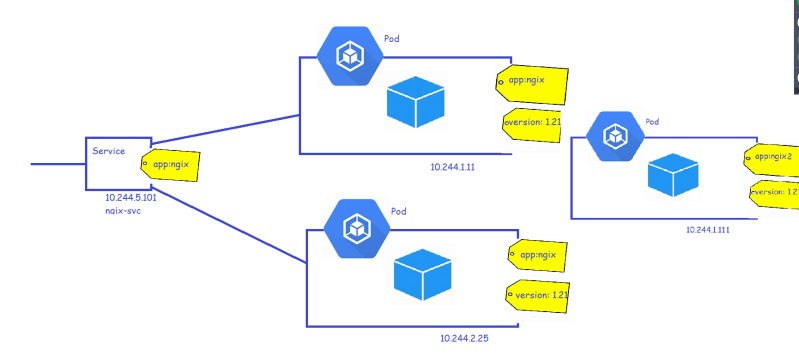
* Annotations provide a place to store additional metadata for kubernetes objects with sole purpose of assisting tools and libraries

metadata:

annotatations:

qt.com/icon-url: "https://qt.com/icon.png"

## Service Discovery

* Let’s assume we have create a replica set of k8s with two pods
* Each pod will have labels and will have unique IP address
* How can these pods be accessed by other pods in cluster and externally 
* Using Pod IP is not sensible even with in the cluster as Pods can be added or removed either dynamically or manually
* Kubernetes has service discovery and it supports DNS and load balancing
* Service Object:
  + K8s service discovery starts with Service Object 
  + Because the cluster ip (ip address given to service ) is virtual and it is appropriate to give it a DNS address
  + Service can be accessed also by a DNS name and k8s provides a DNS service exposed to pods running in cluster
  + The DNS for service will be <service-name>.<namespace>.<svc>.<cluster.local>
  + For the service shown in image the DNS name would be ngix-svc.default.svc.cluster.local
* Let’s create a nginx service with cluster ip [Refer Here](https://github.com/asquarezone/DockerZone/commit/671601b2d59ab00d1019e13be8c3e2070d7052bb)

---

apiVersion: v1

kind: Service

metadata:

name: nginx-svc

spec:

selector:

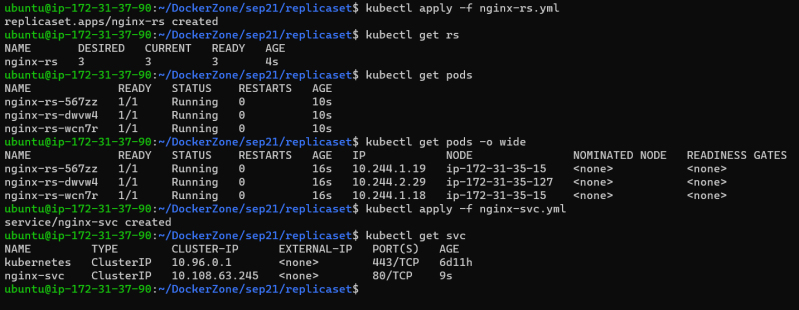
app: nginx

ports:

- protocol: TCP

targetPort: 80

port: 80

* Create replica set and service 
* Service Types: [Refer Here](https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services-service-types)
* [Refer Here](https://github.com/asquarezone/DockerZone/commit/45118aa30149a2b05665233cc07e0cf8ffda11b1) for the changes done to add node port
* We will look into load balancers in AKS and EKS

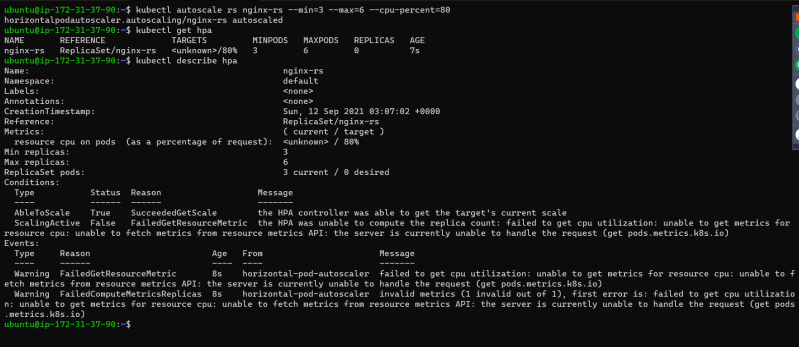
## Scaling Replica Sets

* Imperatively scale

kubectl scale replicaset <rs-name> --replicas=<desired count>

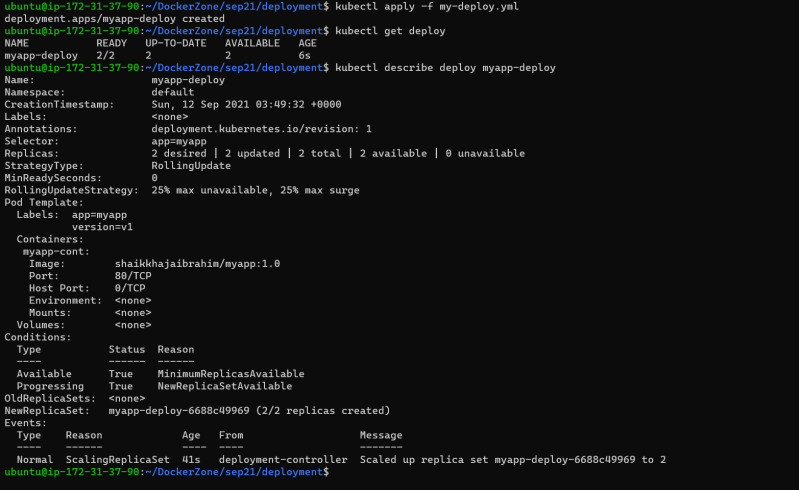
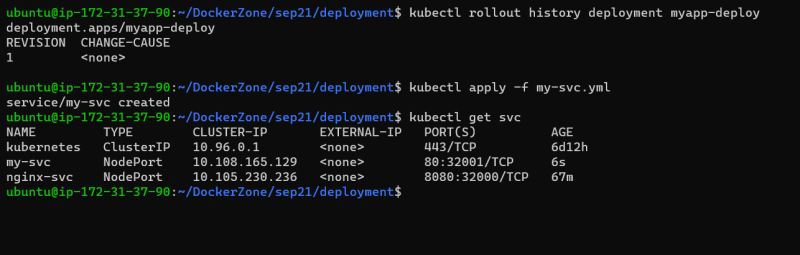
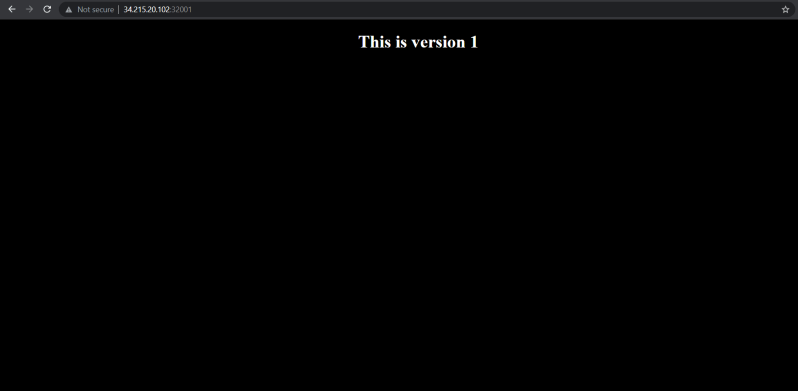
* Declaratively scale: Change the yaml field repli case in replica set manifest and apply
* Auto scaling a Replica set: k8s can handle scaling automatically based on CPU and memory consumptions using Horizontal Pod Auto Scaling (HPA)
* Now let’s try to autoscale rs using cpu

kubectl autoscale rs nginx-rs --min=3 --max=6 --cpu-percent=80

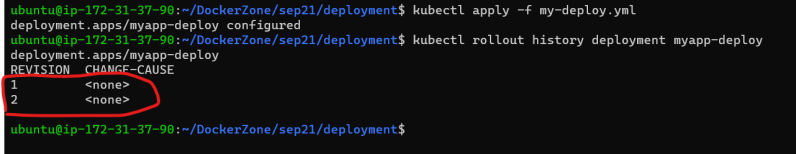


* Manifest to HPA also can be written [Refer Here](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.21/#horizontalpodautoscaler-v1-autoscaling)

## Deployment

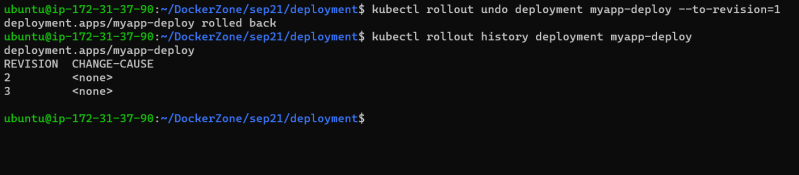
* We can run our application in pods with auto scaling configured and expose our application to the outside world using service
* However they do little to help you manage the daily or weekly releases of your applications
* The Deployment object exists to manage the release of new versions.
* Deployments enable you to easily move from one version of your application/code to the next. This is referred as rollout process.
* In some case, the new rollout might lead to errors/application issues and we would like to go back to previous versions. K8s deployment covers this scenario as well as the deployments in k8s support roll-backs (undo rollouts)
* Let’s create a Deployment for an application [Refer Here](https://github.com/asquarezone/DockerZone/commit/f3342e17edfd61a38b9867b671025db2a15e6f41)   
* The deployment manifest is changed to new version [Refer Here](https://github.com/asquarezone/DockerZone/commit/087ca5803c9a8dfdf90c11e1a990c56c49ec911e)
* Now let’s apply the deployment with –record flag

kubectl apply -f my-deploy.yaml --record="version 2"



* Now let’s assume we need to rollback to revision 1

kubectl rollout undo deployment myapp-deploy --to-revision=1



* Deployment Strategies
  + Recreate:
    - It simply updates the Replica Set it manages to use the new image and terminates all the Pods associated with deployment.
    - The RS notices it no longer has any replicas and recreate all Pods with new changes
    - This is simple and fast but has one major drawback i.e. this strategy will have some downtime
  + Rolling Update:
    - This recommended strategy
    - This deployment has max Surge and max Unavailable fields
    - Max Surge is the maximum number of new Pods to be created above the desired number of pods and can be expressed in numbers or percentage
    - Max Unavailable: He maximum number of pods that can be unavailable during the update.
    - [Refer Here](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.21/#deploymentstrategy-v1-apps) for the documentation
* [Refer Here](https://github.com/asquarezone/DockerZone/commit/fbb53c330f86236994f046db69aa75b246758ca6) to view the deployment manifest with strategy

### Share this:

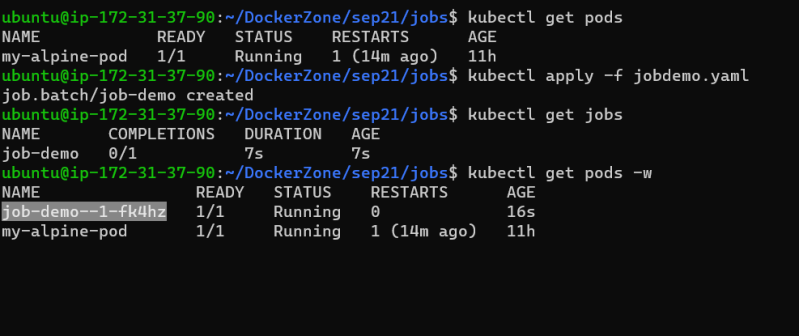
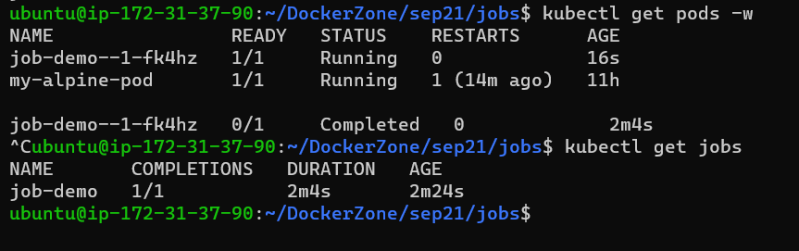
SEPTEMBER 13, 2021

# DevOps Classroom Series (Evg) – 12/Sept/2021

## Daemon Sets

* A Daemon set ensures a copy of Pod is running across set of nodes in k8s cluster
* Daemon Sets are used to deploy system daemons such as log collectors and monitoring agents which typically must run on every node.
* [Refer Here](https://github.com/asquarezone/DockerZone/commit/dc3c5a5caee6d8b1c7d392ad1764694076715b8c) for the change set

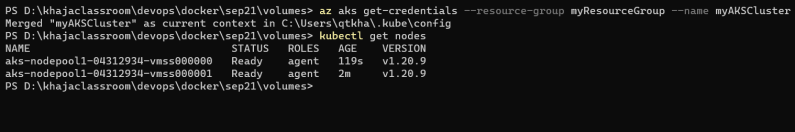
## Job

* A job is supervisor in k8s that can be used to manage pods to run some script/task and exit gracefully
* The pods created by the job are not deleted following the completion of job, the pods run to completion and stay with completed status
* To test this lets create an pod with alpine container which sleeps for 2m and restart Policy On Failure
* [Refer Here](https://github.com/asquarezone/DockerZone/commit/2ac2ecd74987655c49b5c9b5c8eca52a1f3ecc5b) to the change set containing dummy job  
* Exercise: Write a cron job spec which runs alpine pod for 2 minutes every hour (every 1/2 hour) using cron job spec
  + [Refer Here](https://crontab.guru/) for cron expression
  + [Refer Here](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.21/#cronjob-v1-batch) for cron job spec

## k8S on Cloud Platforms

* Azure Supports k8s as service with a Service offering called as Azure Kubernetes Service (AKS)
* AWS Supports k8s as service with a Service offering called as Elastic Kubernetes Services (EKS)
* GCP Support k8s as Service with Service offering called as Google Kuberentes Engine (GKE)
* The basic approach for any cloud based offering of k8s as Service is
  + The master node is by default highly available and is managed by cloud provider
  + We need not install k8s components on nodes cloud providers will take care of installations.
  + Every Cloud has its version of Networking component (CNI) implemented.

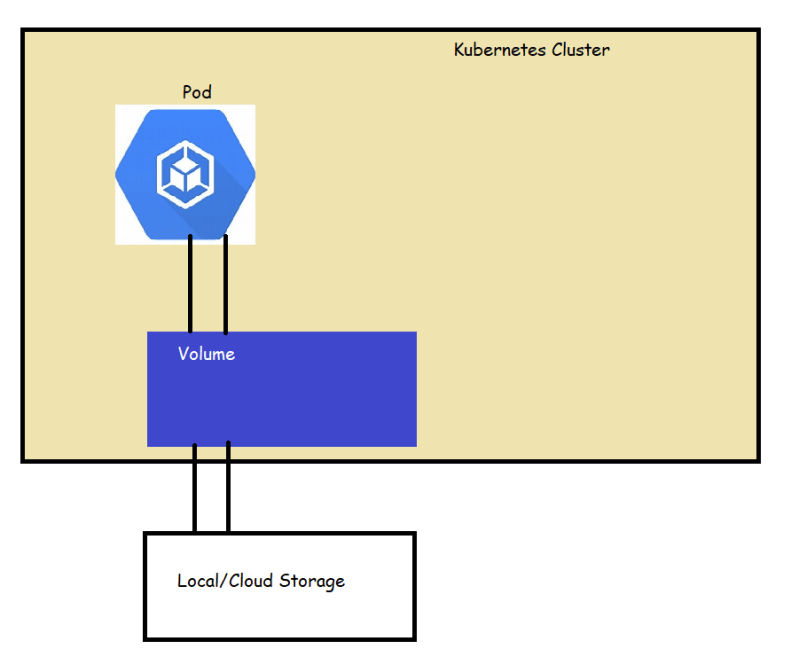
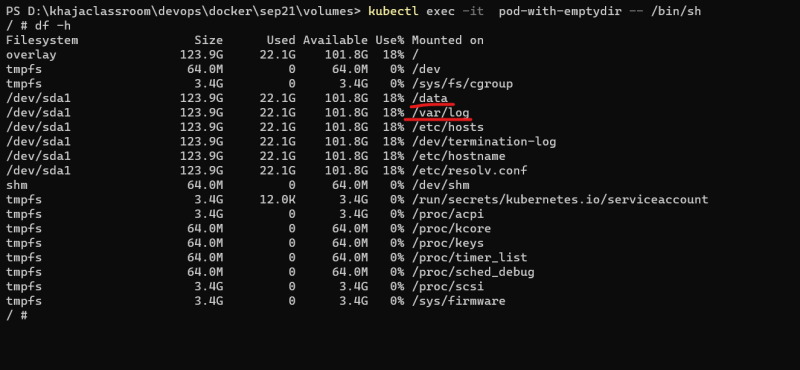
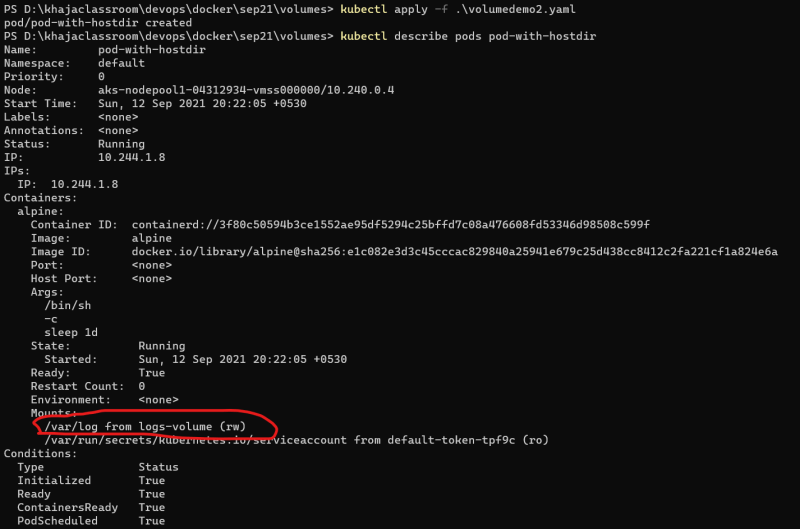
## Azure Kubernetes Services

* Launching AKS [Refer Here](https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough)
* In this case we can install kubectl on your laptop/workstation
  + kubectl installation [Refer Here](https://kubernetes.io/docs/tasks/tools/#kubectl)
* Azure CLI installed on local machine
* Follow the steps mentioned in Launching AKS 

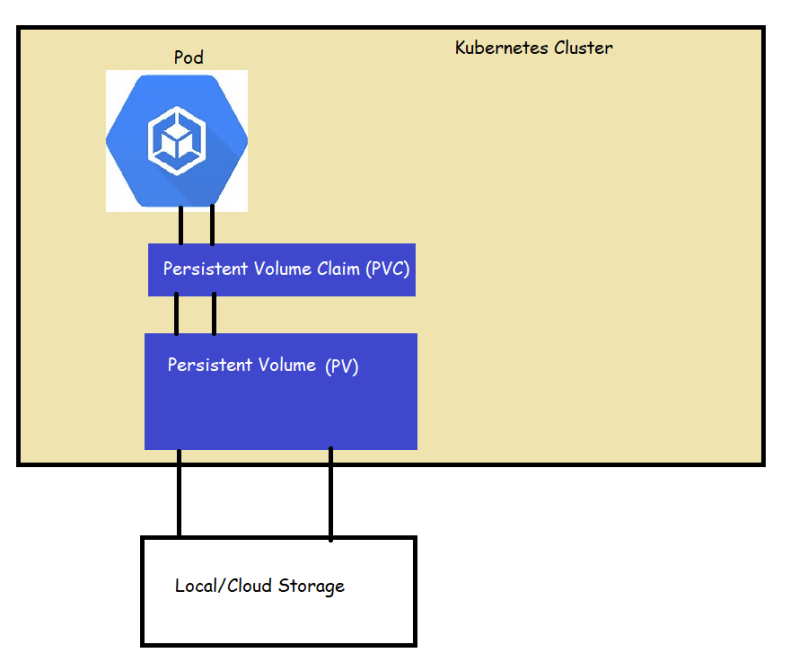
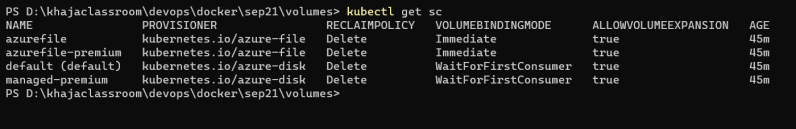
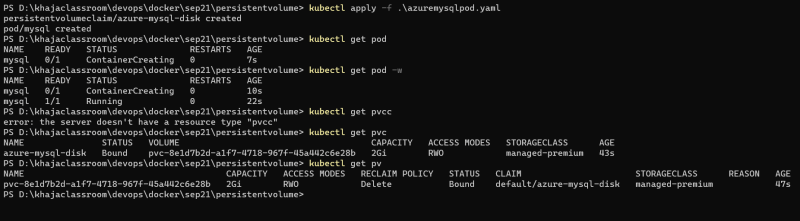
## Problem with Storage

* We have learned that container store the data in read/write layer which gets deleted when the conainer is deleted.
* So in the case of docker we have used volumes
* Now, In k8s also we need to look out for a solution as we are dealing with multiple nodes as well
* Kubernetes has two major solutions
  + Volumes
  + Persistent Volumes

## Kubernetes Volumes

* Volumes are storage abstractions for Pod application 
* Lifetime of K8s Volume is same as the Pod that uses.
* Even if the containers in the pod restart the same volume will be used by containers and Data will not be lost, but when pod is deleted or terminate volume will also be deleted
* Types of volumes
  + Empty Dir
  + Host Path
  + We also have other cloud based volumes [Refer Here](https://kubernetes.io/docs/concepts/storage/volumes/#volume-types)
* Let’s create a sample pod with empty dir volumes [Refer Here](https://github.com/asquarezone/DockerZone/commit/394a00bb1fe1307e19151300202a400a6b650a48) 
* Let’s login into container and then look it mounts 
* Let’s create a host path [Refer Here](https://github.com/asquarezone/DockerZone/commit/8a92cc670df65a17c3d50745043dbe8136a06e07) 

## Persistent Volumes

* K8s supports persistent storage in the form of Persistent Volumes (PV)
* PV is k8s object that represents a block of storage in cluster which can be provisioned manually or we can dynamically provision
* PV is a cluster resource. Life cycle of PV does not depends on Pod but will be alive as long as cluster is. 
* For official docs [Refer Here](https://kubernetes.io/docs/concepts/storage/persistent-volumes/)
* [Refer Here](https://kubernetes.io/docs/concepts/storage/persistent-volumes/#types-of-persistent-volumes) for types of persistent volumes
* Each Persistent Volume Belongs to a certain storage class
* All the cloud based providers support different types of storage classes and they will have default storage classes as well
* In the case of AKS 
* Azure Disk
  + Dynamic [Refer Here](https://docs.microsoft.com/en-us/azure/aks/azure-disks-dynamic-pv)
  + Static [Refer Here](https://docs.microsoft.com/en-us/azure/aks/azure-disk-volume)
* Azure Files:
  + Dynamic [Refer Here](https://docs.microsoft.com/en-us/azure/aks/azure-files-dynamic-pv)
  + Static [Refer Here](https://docs.microsoft.com/en-us/azure/aks/azure-files-volume)
* Let’s create a MySQL volume for a MySQL pod with Persistent Volume Claim (Dynamic provisioning) [Refer Here](https://github.com/asquarezone/DockerZone/commit/d801cddcd31984bfe9dc39b9945b9b732492a8fd) 

### Share this:

* [Click to share on Twitter (Opens in new window)](https://directdevops.blog/2021/09/13/devops-classroom-series-evg-12-sept-2021/?share=twitter&nb=1)
* [Click to share on Facebook (Opens in new window)](https://directdevops.blog/2021/09/13/devops-classroom-series-evg-12-sept-2021/?share=facebook&nb=1)
* [Click to share on WhatsApp (Opens in new window)](https://directdevops.blog/2021/09/13/devops-classroom-series-evg-12-sept-2021/?share=jetpack-whatsapp&nb=1)

### Like this:

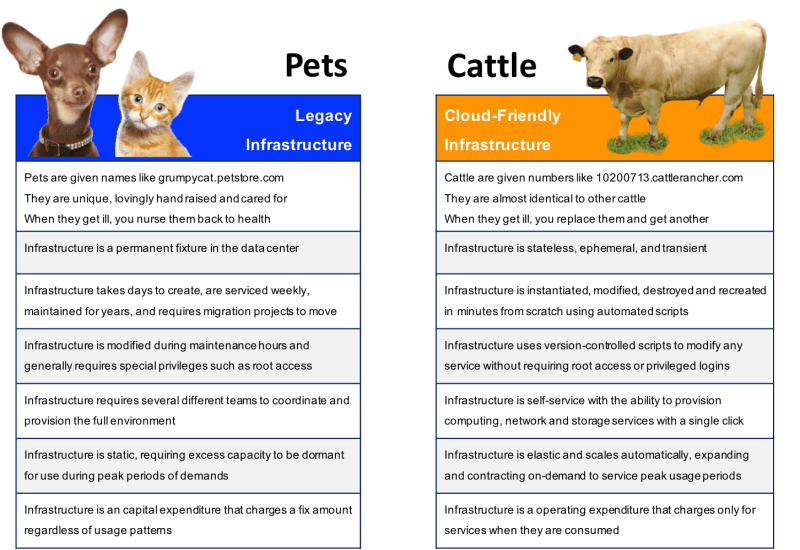
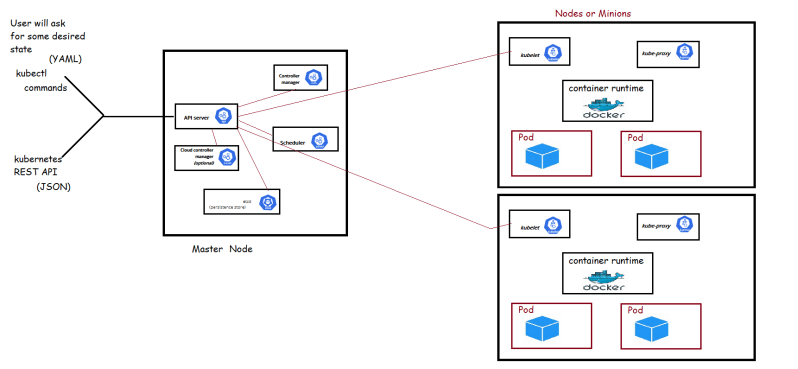
### ==================================@@@@@@@@@@@@@@@@@My K8S class completed….. Now last batch data my Khaja

### -------------------------------------------------------------------====================================

APRIL 4, 2021

# DevOps Classroom Series – 04/Apr/2021

### Kubernetes (k8s)

* k8’s is a portable, extensible, open source platform for managing containerized workloads and services.
* It facilitates both declarative configuration and automation
* k8s provides
  + Service discovery and load balancing
  + Storage Orchestration
  + Automated rollouts and rollbacks
  + Automatic bin packing
  + Self-healing
  + Secret and configuration management
* Kubernetes is very popular container orchestration platform
* To know more about kubernetes [Refer Here](https://kubernetes.io/docs/home/)
* All major cloud providers such as AWS, GCP, Azure have native kubernetes support
* Cattle vs Pets
  + Article [Refer Here](https://joachim8675309.medium.com/devops-concepts-pets-vs-cattle-2380b5aab313) 
* [Refer Here](https://kubernetes.io/docs/concepts/overview/components/) for official documentation on kubernetes components
* Basic components of k8s 

APRIL 6, 2021

# DevOps Classroom Series – 06/Apr/2021

### Installing Kubernetes

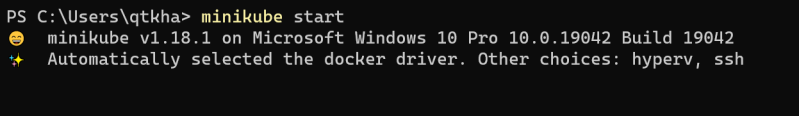
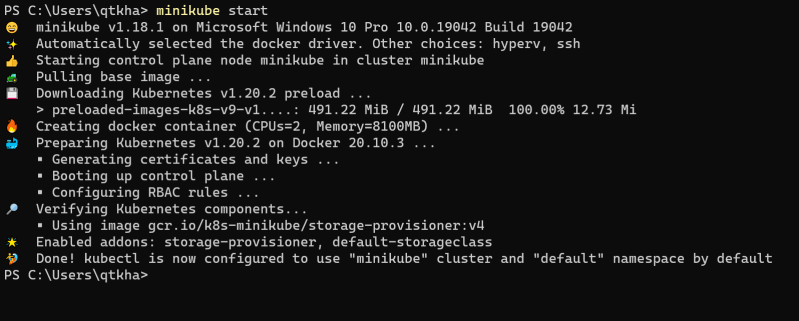
* K8s can be installed on a single node to set up the developer environments.
  + minikube
* On production environments we would be using a multi node setup
  + For IOT / ARM processers => K3s (Lightweight kubernetes)
  + Kubeadm (multi node setup on Linux workloads)
  + Kops [Refer Here](https://kubernetes.io/docs/setup/production-environment/tools/kops/) for installing k8s on AWS
  + kubespray [Refer Here](https://kubernetes.io/docs/setup/production-environment/tools/kubespray/)
  + Native Kubernetes on Cloud
    - AKS (Azure Kubernetes Services)
    - GKE (Google Kubernetes Engine)
    - EKS (Elastic Kubernetes Services on AWS)
* Let’s try to setup
  + minikube
  + kubeadm

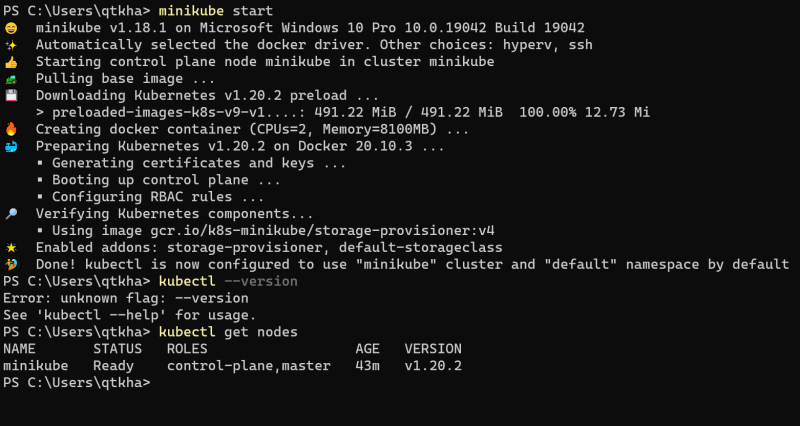
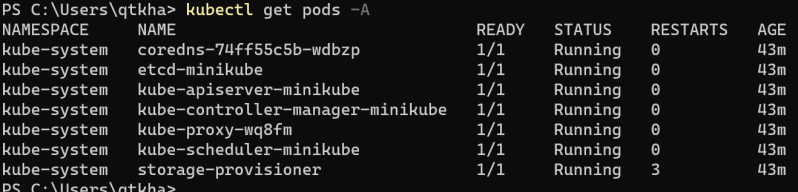
#### Minikube

* This is a tool to set up a single node cluster and suitable for local environments.
* To setup minikube [Refer Here](https://minikube.sigs.k8s.io/docs/start/)
* Commands

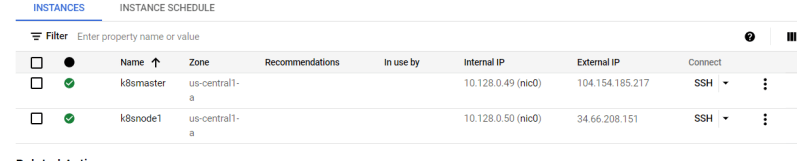
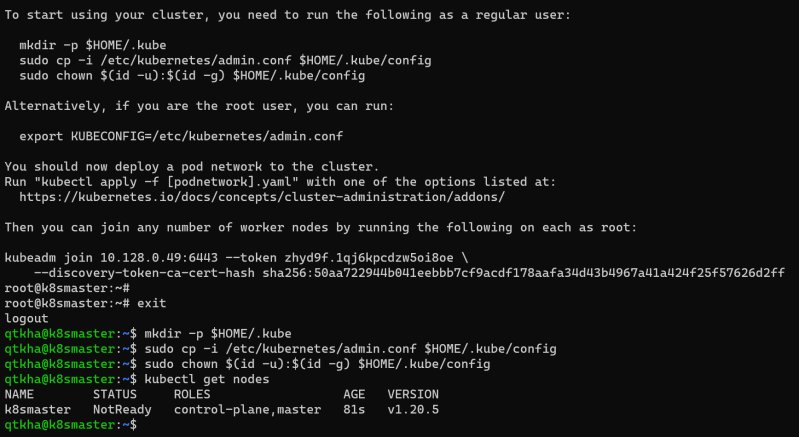
choco install minikube -y

minikube start

* Check the nodes 
* Check all the pods 

### Kubeadm

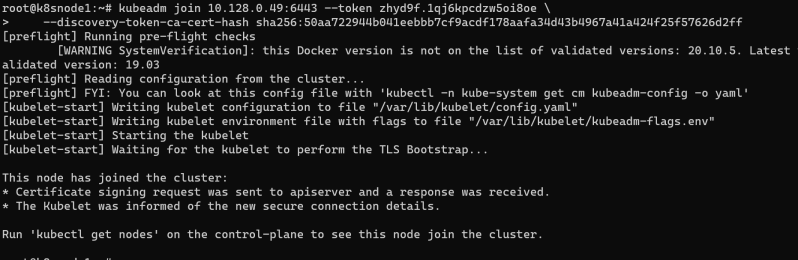
* In this article we will be creating a kubernetes cluster on multiple linux nodes using kubeadm [Refer Here](https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/)
* We would be creating a 3 node kubernetes cluster 
* Let’s follow the installation guide [Refer Here](https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/)
* Install container runtime on the nodes. Let’s use docker container [Refer Here](https://kubernetes.io/docs/setup/production-environment/container-runtimes/#docker)
* Now install kubeadm, kubectl and kubelet by following instructions [Refer Here](https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/#installing-kubeadm-kubelet-and-kubectl) 
* Now let’s install pod network

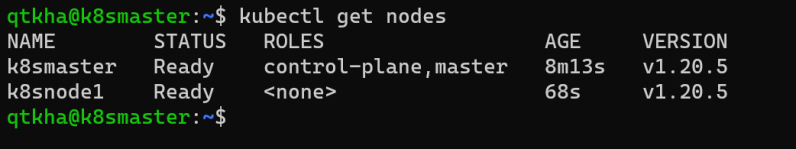
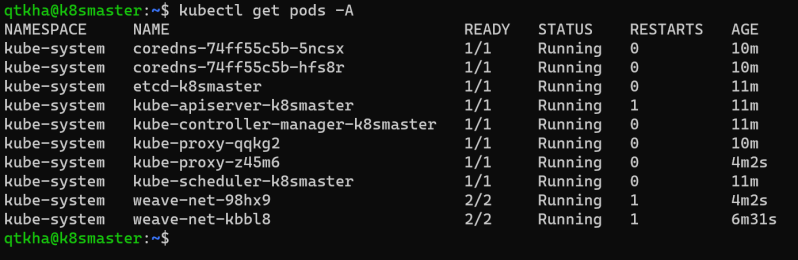
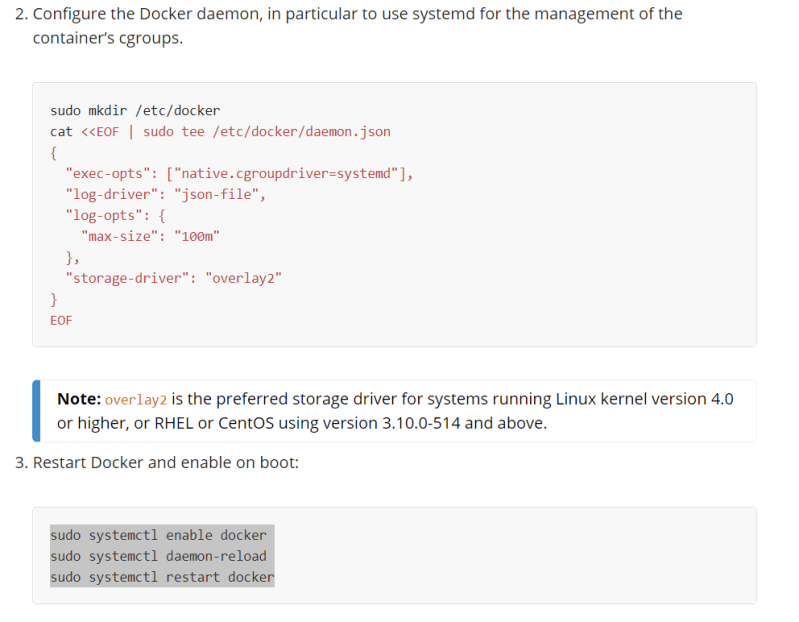
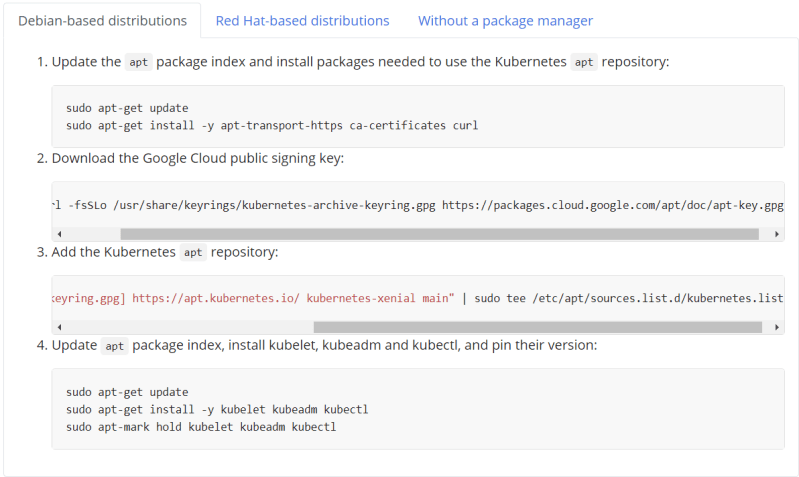
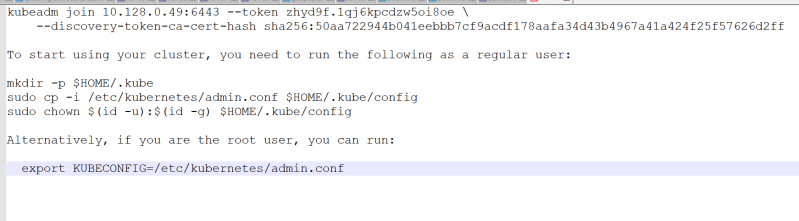
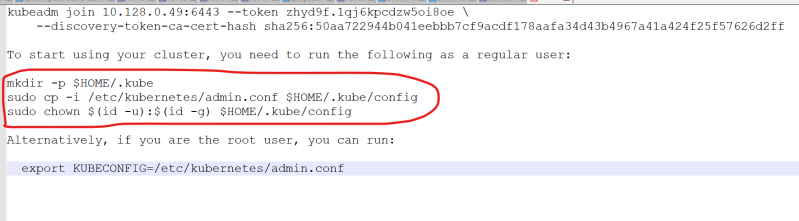
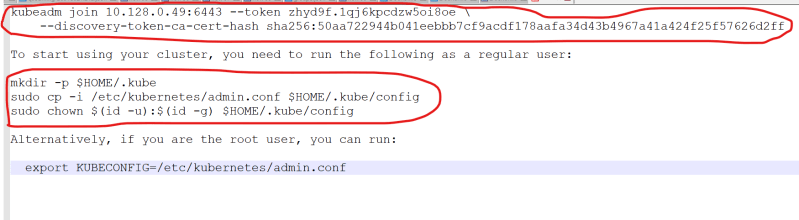
kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"



* Now let’xxxxxxxxxxxxxxxxxxxxxxxs login into the node become a root user and execute the kubeadm join command

kubeadm join 10.128.0.49:6443 --token <somtoken>



* Now login into k8s master and execute kubectl get nodes  
* Summary of installation
  + Install docker on the nodes (both) and execute the steps mentioned in [Refer Here](https://kubernetes.io/docs/setup/production-environment/container-runtimes/#docker) 
  + Install the kubeadm, kubelet and kubectl on both nodes [Refer Here](https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/#installing-kubeadm-kubelet-and-kubectl) 
  + Now initialize the kubeadm by loggin in as root user on master node
* kubeadm init
  + Copy the commands to configure the kubectl and join command 
  + Now become the non root user and execute the following 
  + Now install a pod network on master node and verify if the node is in ready state by executing kubectl get nodes
* kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"
  + Now login into node and become root user (sudo -i) and execute the join command 
  + Now login into master node and execute kubectl get nodes to see the status of both nodes

### Share this:

APRIL 7, 2021

# DevOps Classroom Series – 07/Apr/2021

### Kubernetes Objects

* K8s objects are persistent entities of k8s system
* k8s object is "record of intent".
* Every k8s object will have an Object Specification (Spec) and object status
* The specificiation is about what we want and is regarded as desired state and the status is what is the actual state of object in k8s
* We specify object spec in the format of .yaml files and we get the status back after we apply the spec
* Lets take an example of the Pod spec

---

apiVersion: v1

kind: Pod

metadata:

name: hello-pod

labels:

app: hello-pod

spec:

containers:

- name: web

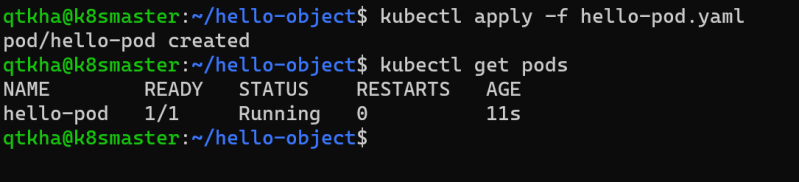
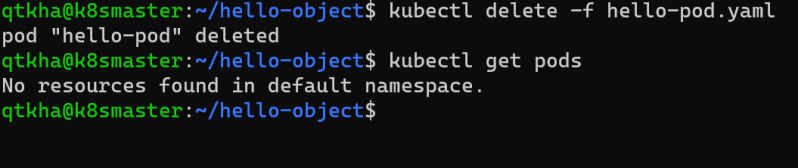
image: httpd

ports:

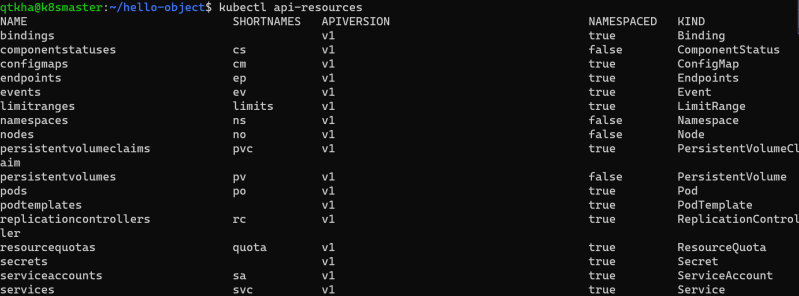
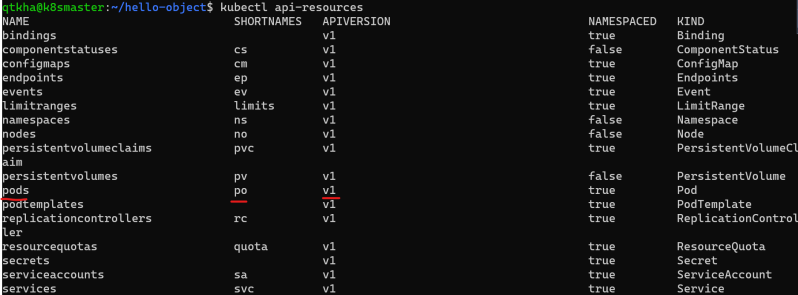
- name: web

containerPort: 80

protocol: TCP

* Now apply the spec using kubectl apply -f and view the status of the object 
* Now delete the spec using kubectl delete -f 

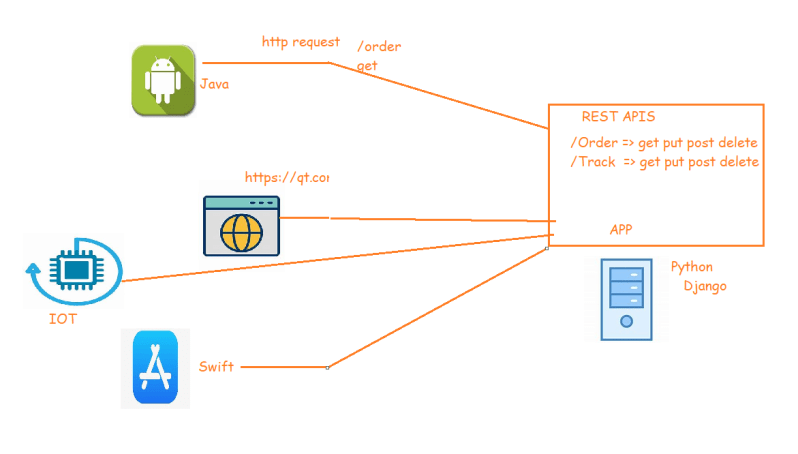
#### **Kubernetes API**

* The core of the k8s control plane is the Api Server. This API server exposes HTTP API that lets enduser, different parts of your cluster and external components communicate with one another
* K8s stores the serialized state of objects by writing them to etcd
* To express the desired state we would be creating the specifications of the kubernetes resources
  + Pod
  + Container
  + Ephemeral Containers
  + Replication Controllers
  + Replica Sets
  + Deployments
  + StatefulSets
  + DaemonSet
  + Job
  + CronJob
  + HorizontalPodAutoScaler
* For the resources you can execute kubectl api-resources and then view the list of resources.
  + Resources will have api-version, kind, Namespaced and some of the resources will have shortnames 
* To create this resources we will be using kubectl and specify the spec in yaml file and then apply 
* API groups and versioning:
  + API Groups make it easier to extend the k8s api. The API group is specified in the REST path and in the apiVersion field of serialized object
  + There are several api groups
    - The core group is found at REST path /api/v1. This is simply written as v1
    - The named group: /api/$GROUP\_NAME/$VERSION example: apiVersion: batch/v1
  + [Refer Here](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.20/#-strong-api-groups-strong-) for API Groups

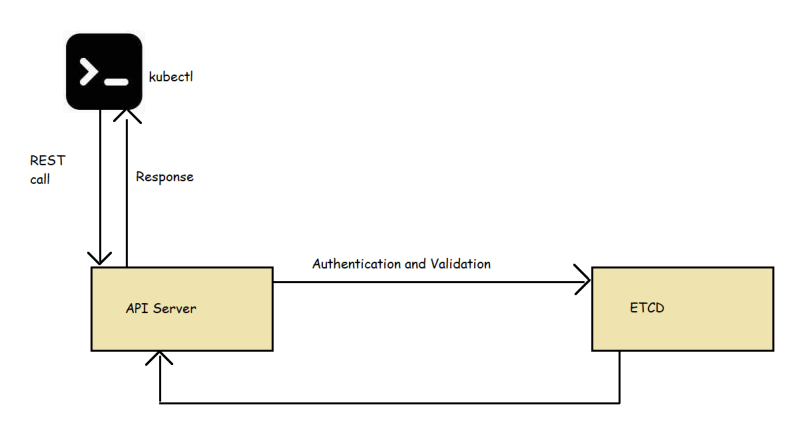
### kubectl

* This is a command line utility for interacting and performing various operation on k8s.
* Kubectl has two ways of managing your cluster
  + imperative:
    - We create objects by executing kubectl commands
  + declarative:
    - We create objects by specifying the desired state as specification in YAML format

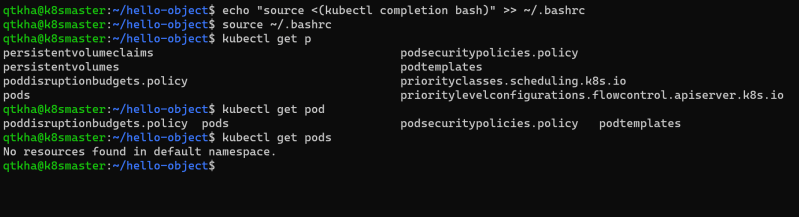
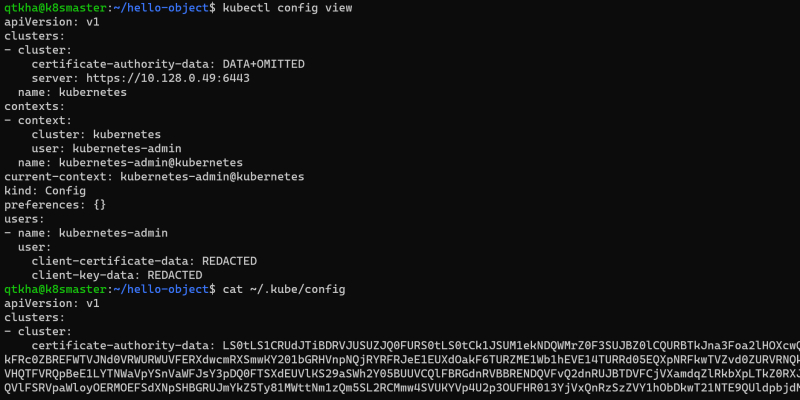
#### **HTTP Protocol and REST API**



#### **How kubectl Communicates with k8s**

* The API Server manages communications in k8s
* To achieve this , API-server implements the RESTful API over http and https to perform CRUD operations to populate and modify k8s API objects (pods, services, deployments) 

#### **Kubectl cheatsheet**

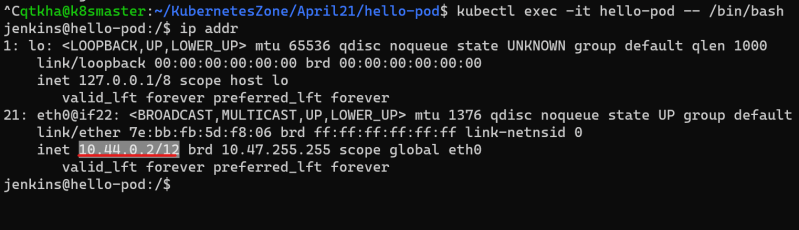
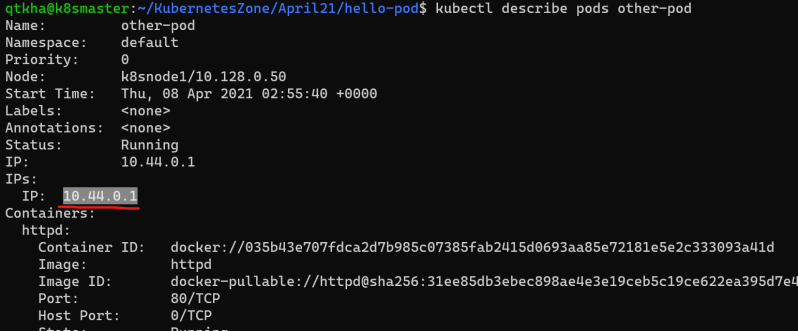
* [Refer Here](https://kubernetes.io/docs/reference/kubectl/cheatsheet/) for the official kubectl cheatsheet
* Now lets setup autocompletion 
* kubectl configuration 

### Share this:

APRIL 8, 2021

# DevOps Classroom Series – 08/Apr/2021

### Pod

* This is smallest deployable units of computing in k8s
* A Pod consists of one or more containers with shared network and storage resources
* Each Pod will recieve a unique ip address
* To create any object in k8s we need to create a template
* To write the spec navigate to api reference [Refer Here](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.19/)
* [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/2cd30bd9cb16740c2731c231cd8e4caee10744c7) for the pods specs created
* Now lets try to apply the pod spec created 
* Each pod will get one ip address  

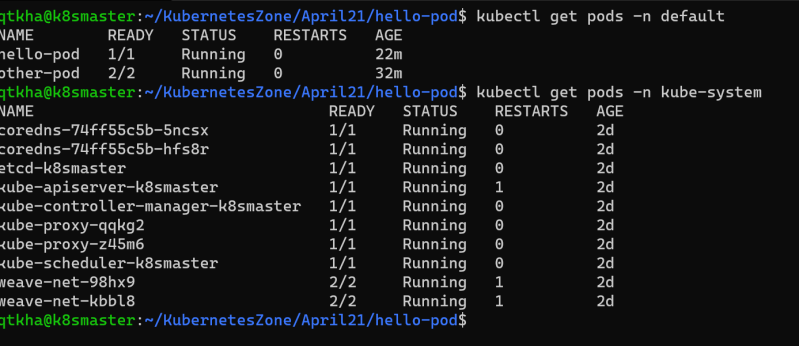
#### **kubernetes namespace**

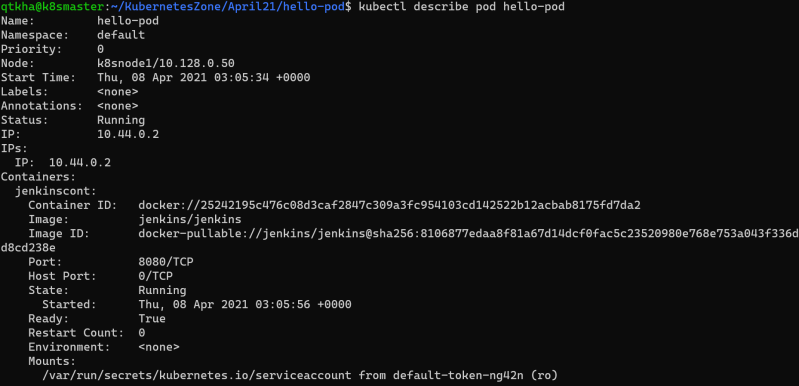
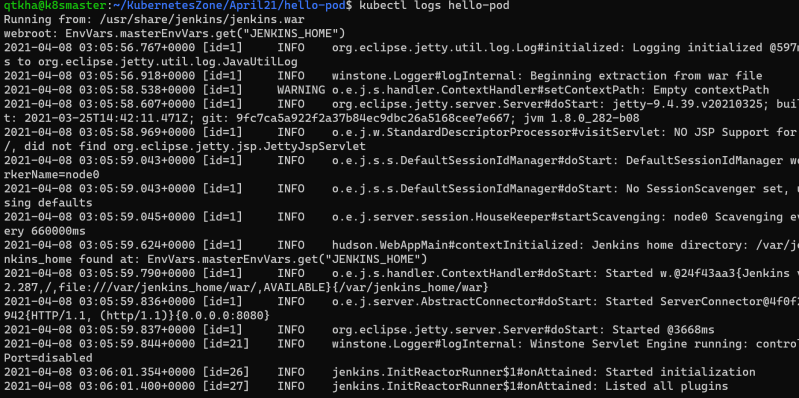
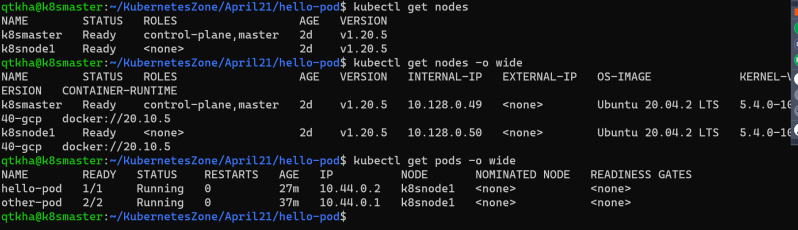
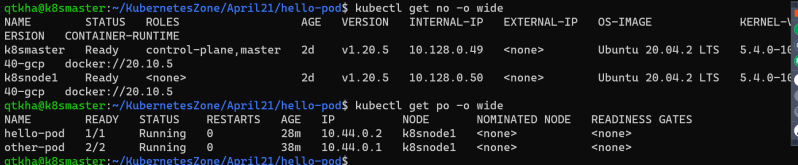
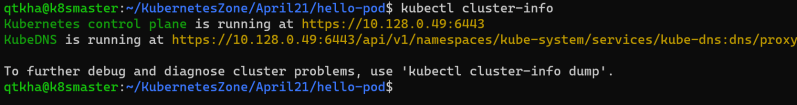
* K8s supports multiple virtual cluster backed by same physical cluster and these virtual clusters are called as namespace
* In k8s for any object we have two kinds of scopes
  + cluster scope
  + namespace scope

#### **Commonly used kubectl commands**

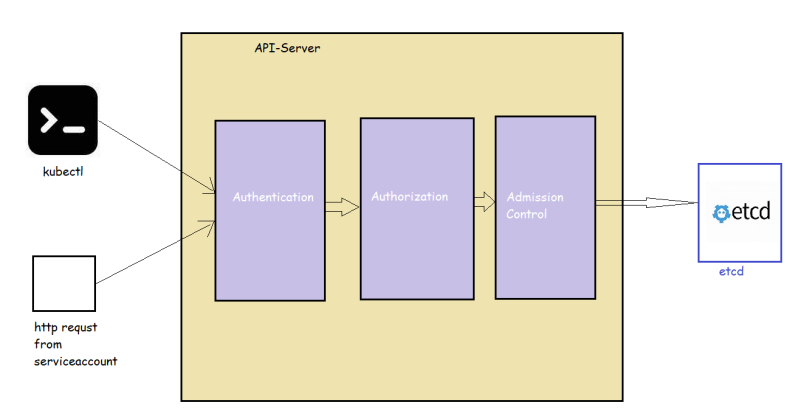
* get object: In the resources which are scoped to namespace we can use -n to specify the namespace

kubectl get pods -n default

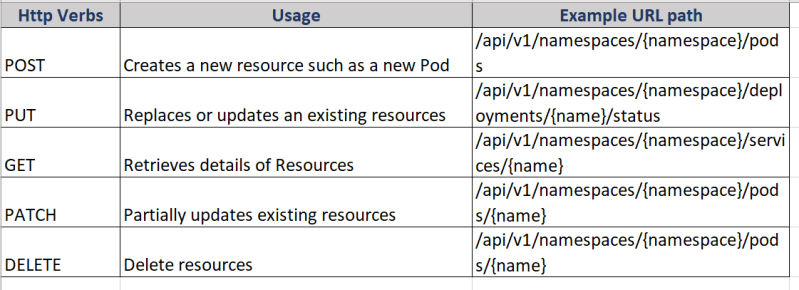
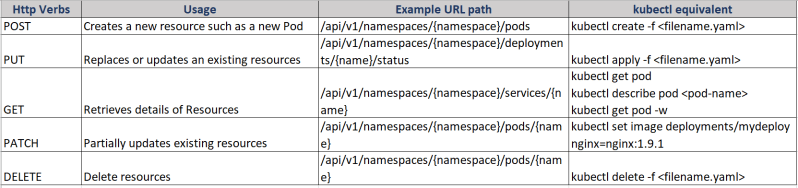
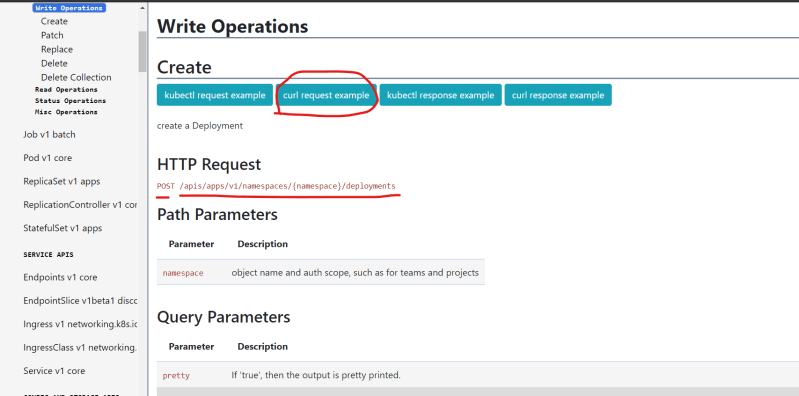
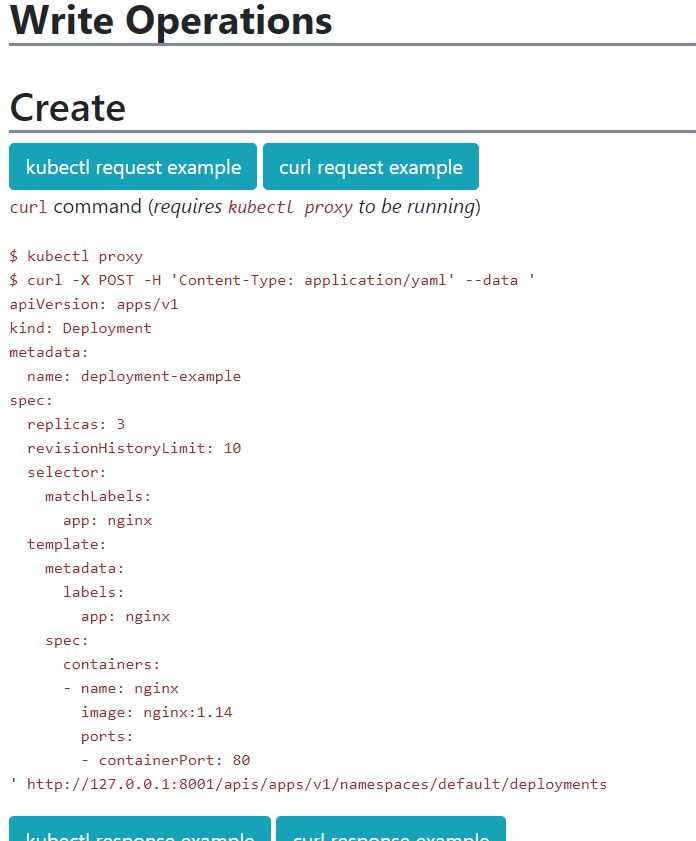


* describe object-type object-name: 
* logs object-name: 
* apply -f filename.yaml
* Other Commands 
* We can use short names 
* To get to know the information of cluster 

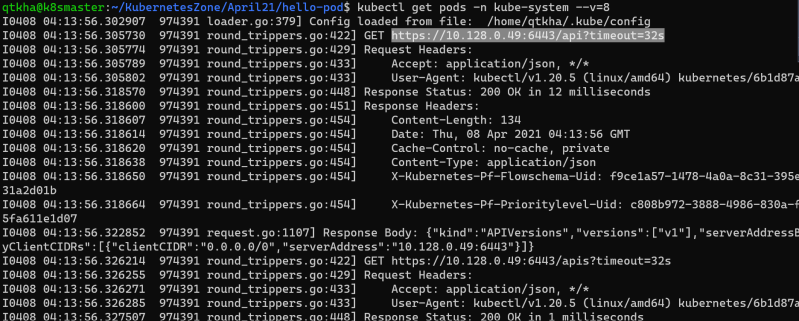
#### **Kubernetes HTTP Request Flow**

* HTTP Request flow 
* Restful api [Refer Here](https://restfulapi.net/)

#### **The K8s API**

* K8s API uses JSON over http for its requests and responses.
* k8s api allows clients to create, update, delete or read the description of object via standard http methods    
* Trace the HTTP requests: To trace http request we will be using kubectl get pods -n kube-system. To this we will be adding verbose level –v=8

kubectl get pods -n kube-system --v=8

 \*

### Share this:

APRIL 9, 2021

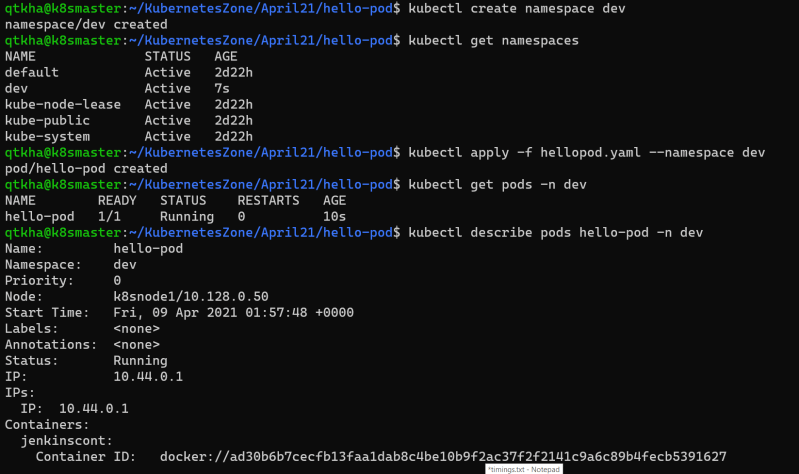
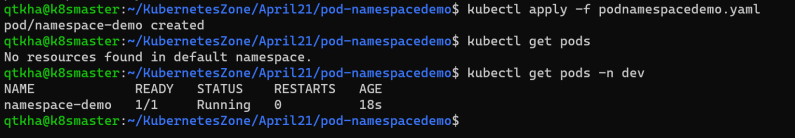
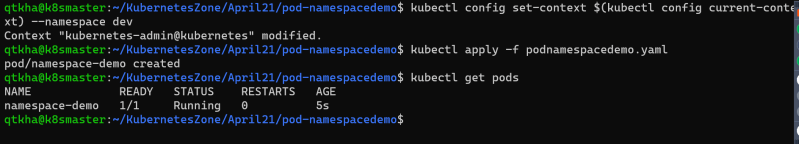
# DevOps Classroom Series – 09/Apr/2021

### Pods Contd

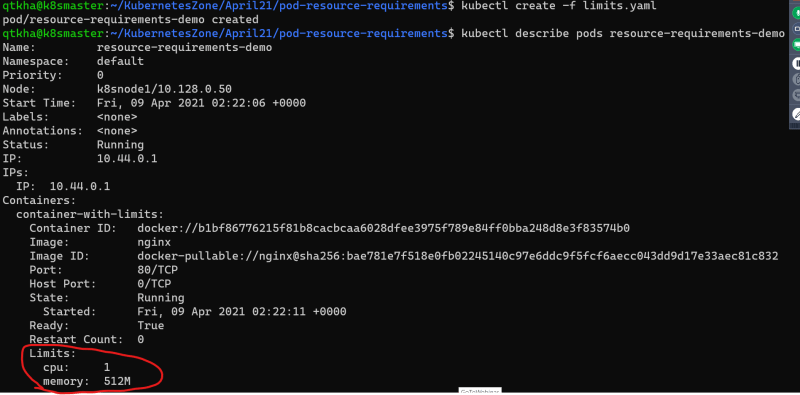
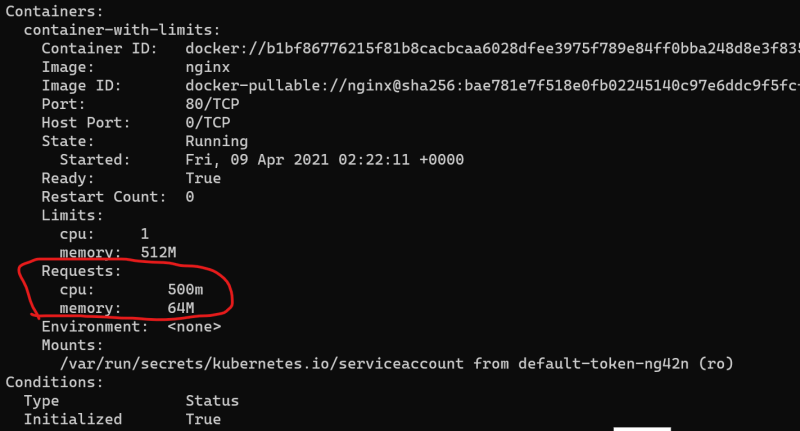
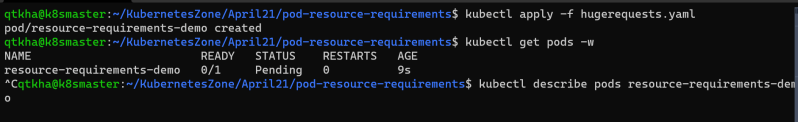
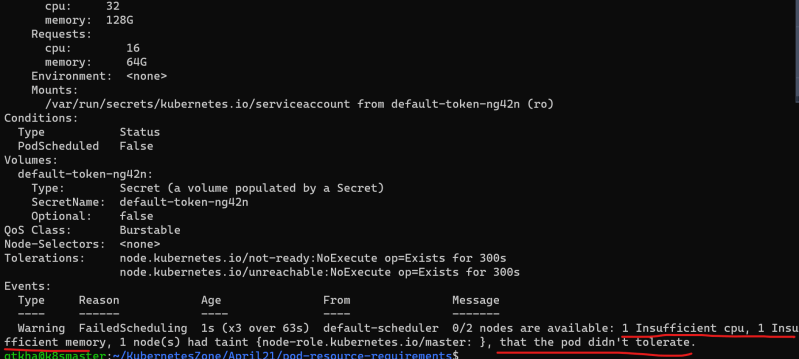
* kubectl port-forward was not working. To fix this we need to use the –address

kubectl port-forward <pod-name> <portmapping> --address <private ip>

kubectl port-forward hello-pod 8081:8080 --address 10.128.0.49

* Our Pod configuration is divided into four components
  + **Api Version**: version of k8s api
  + **kind**: This is kind of kubernetes object in our case it is Pod
  + **metadata**: metadata or information that uniquely identifies the object which we are creating
  + **spec**: Specification of our pod with container name image, volume and resource requests
* Let’s create a namespace and run a pod within the namespace 
* [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/ac9849cf3d6a13fbd66d366dd2174fd2c5305136) for the changeset with the namespace demo pod spec 
* To change the default namespace to dev (or any other namespace) 

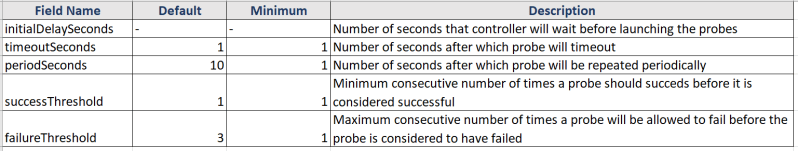
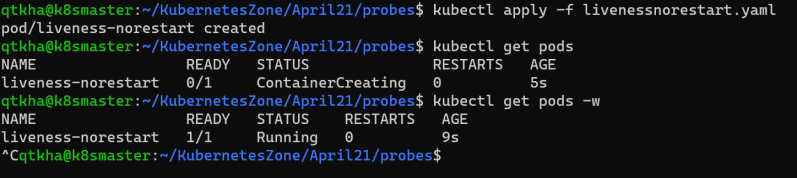
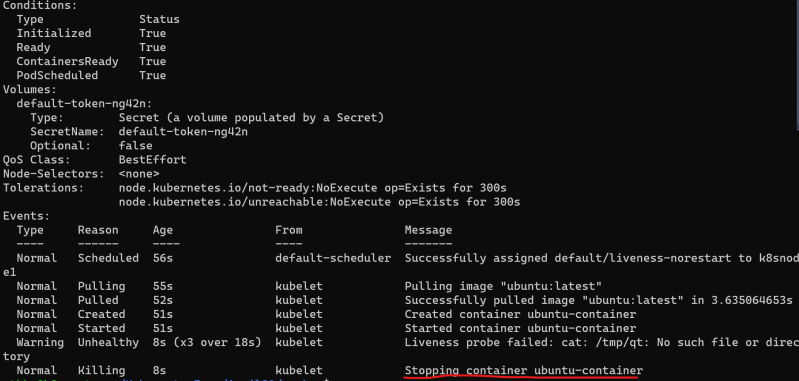
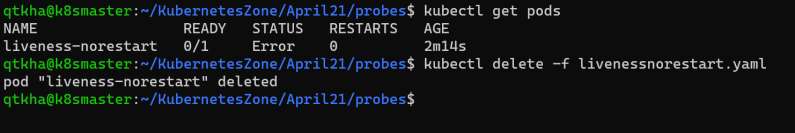
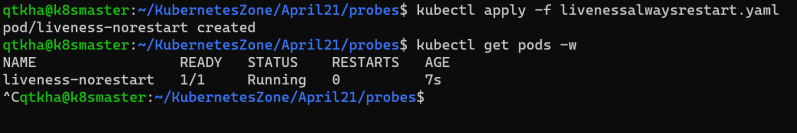
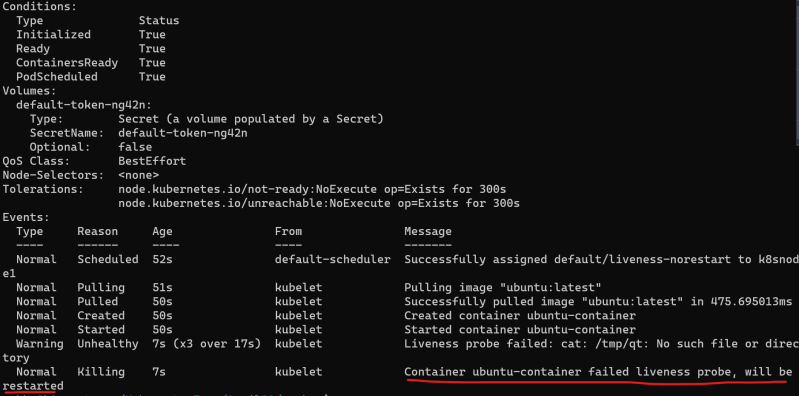
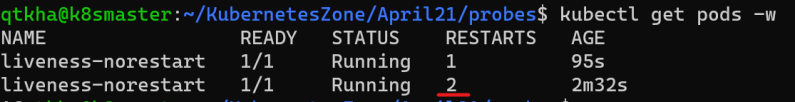
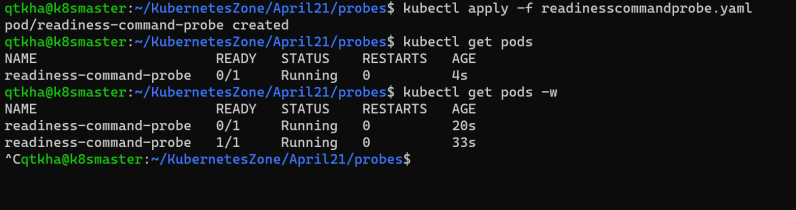
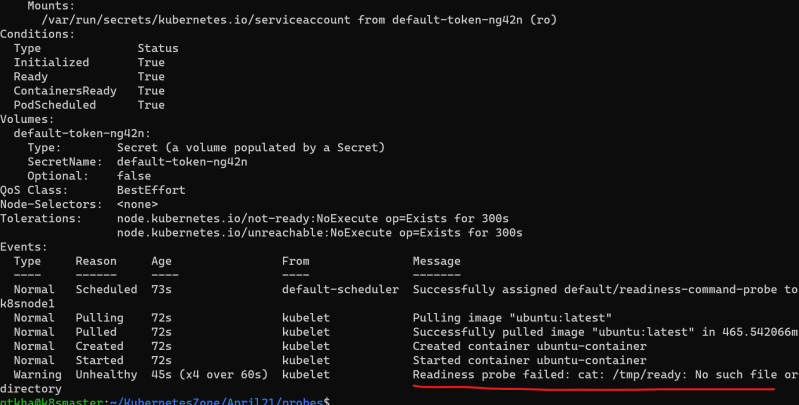
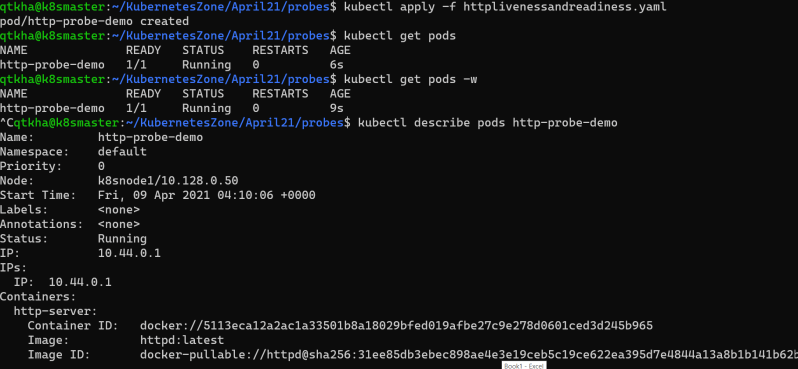
#### **Scenario: Create a Pod running a Container with Resource Requirements**

* [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/ca88b6364aa70cbd9c1b317eac5b25e03a8381a6) for the pod spec yaml  
* Now lets try to add the resource request which are very much huge that we dont have any node suitable for the limits. [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/a8ae0d1b538a17edcd2b7b0164717d65c6b0579d)  

#### **Kubernetes Pod Lifecycle**

* Pod can have the following states
  + Pending: This means the pod has been submitted to the cluster, but the controller hasn’t created all its containers yets. It may be downloading images or waiting for the pod to be scheduled on one of its cluster nodes
  + Running: This state means pod is assigned to one of the cluster nodes and at least one of the containers is either running or is in the process startup
  + Succeeded: This state means the pods has run and all of the container have been terminated with success
  + Failed: This state means that the pod has run & at least one of the containers has terminated with non-zero exit code (failure code)
  + unknown: This means theat the state of the pod could not be found

#### **Probes/Health Checks**

* A probe is a health check that can be configured to check the health of containers running in a Pod. A probe may return the following results
  + Success
  + Failure
  + Unknown
* Types of Probe
  + Liveness Probe: This is used to determine if the particular container is running or not. If a container fails the liveness probe, the controller will try to restart the pod on the same node according to the restart policy configured for the pod
  + Readiness Probe: This is used to determine whether a particular container is ready to receive requests or not. If this fails kubernetes controller will ensure that the pod doesn’t receive any requests. If container specifies a readiness probe, its default state will be Failure until readiness probe succeeds.
* Configuration of probes 
* Implementation of Probes
  + Command Probe: In this probe controller will get the container to execute specific command in order to perform probe on the container.
  + HTTP Request Probe: In this probe controller will send a GET HTTP request to the give address (host and port) to perform probe on the container. We can set the following fields to configure the HTTP Probe
    - host
    - path
    - port
    - http Headers
    - scheme:
  + TCP Socket Probe: In this probe, the controller will try to establish a connection on the given host and the specified port number. We can set the following fields for this probe
    - host
    - port
* RESTART policy:
  + We can specify restart Policy in the pod specification to instruct controller about the conditions to restart the pod. The default value of restart Policy is Always. The possible values are
    - Always: Always restart the pod when it terminates
    - On Failure: Restart the pod only when it terminates with Failure
    - Never: Never restart the pod
* Scenario 1: Create a Pod running a container with a liveness probe and no restart policy
  + [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/33b761baea2e727aa169326c3a4794c9f3568a4a) for the kubernetes manifest file
  + [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/817d519d0732ccb2b3cb7f3548fc122e5e4a2a56) with restart policy added
  + Create the pod 
  + describe the pod  
* Scenario 2: Create a pod running a container with restart Policy Always
  + [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/187bae0cf7836db024e364ca0d2f29c99fdb6a73) for the change set   
* Scenario 3: Create a pod running a container with Readiness Probe
  + [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/c71c537296a23a1a1fcbf100bd7e138665845571) for the change set and [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/ba793c4bdc672df7645a29b249f10f3532f6f459) for the corrections  
* Scenario 4: Create a pod with http Probe for liveness and readiness
  + [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/43b8cc68da7bb725d1ae5553776033f86904729a) for the manifest and [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/262c20af14995b0ed02305222ff3eddd5e4abdf4) for the success threshold fix
  + Apply the manifest and get pods info 
* [Refer Here](https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/) for the official docs on k8s probes

### Share this:

APRIL 10, 2021

# DevOps Classroom Series – 10/Apr/2021

### Labels and Annotations

* Labels are the metadata that contains identifiable information to the kubernetes objects. These are basically key-value pairs attached to the objects such as pods
* Each key must be unique for an object
* Labels would appear in metadata section and the yaml would appear as

metadata:

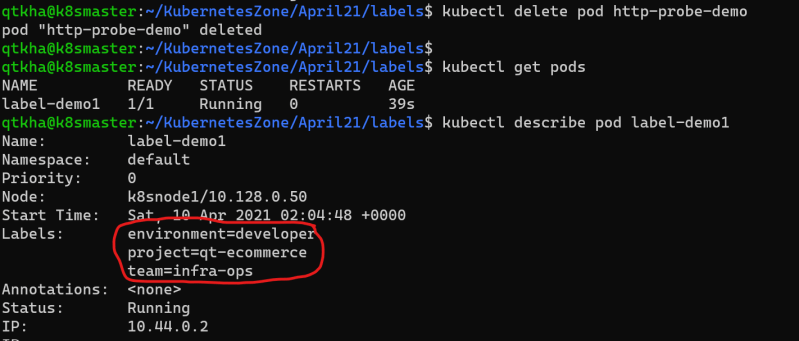
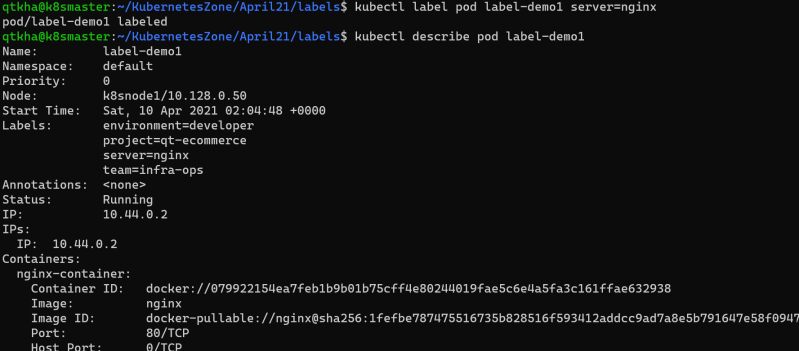
labels:

key1: value1

key2: value2

* Constraints for Labels:
  + Label prefix: This is optional and this must be a DNS subdomain. This cannot be longer thatn 253 characters & cannot contain spaces. The label prefix is always followed by a forwar slash Eg: directdevops.blog/. If no prefix is used the label key is assumed to be private to the user. label prefixes specific to k8s core system kubernetes.io/ and k8s.io/
  + Label name: The label name is required and can be upto 63 characters long.
* Value of the key in k8s is label-prefix/label-name or label-name

#### **Scenario: Lets create a ngnix pod with labels organized by team/project**

* [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/8935697cb0d47f512dd36d7fe1d51200cedba31d) for the k8s manifest and apply the yaml file 
* Add labels to k8s object after creation 
* To change the labels

kubectl label --overwrite pod label-demo1 server=nginx-webserver

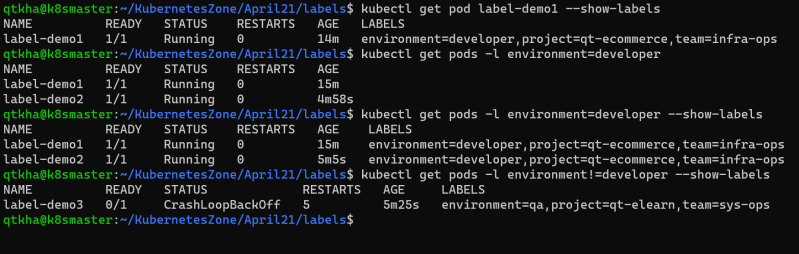
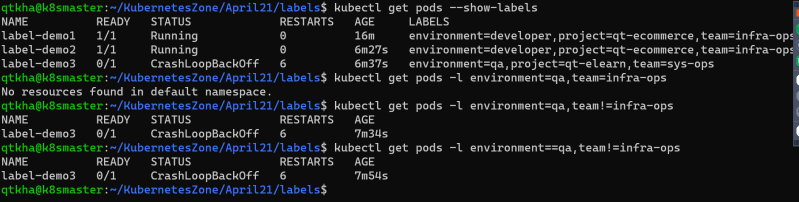
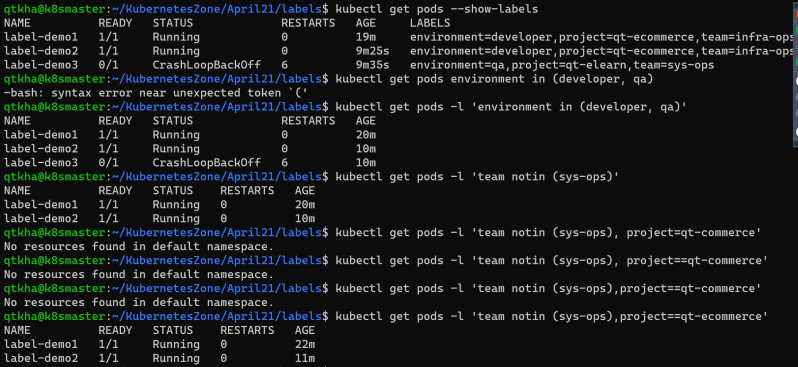
* To Remove the label

kubectl label pod label-demo1 server-

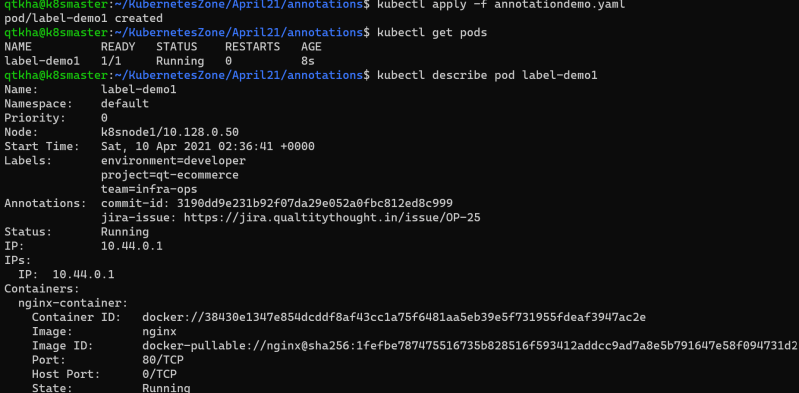
#### **Selecting k8s objects using label selectors**

* Lets create two more pod specs [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/3190dd9e231b92f07da29e052a0fbc812ed8c999) 
* We can query the objects using label selector

kubectl get pods -l {label-selector}

* There are two types of label selectors
  + equality based
  + set-based
* Equality based selectors:
  + In this selector we have three kinds of operators **= == !=**
  + Let’s try to get all the pods with the label environment is equal to developer  
* SET-BASED Selectors
  + In this we have 3 kinds of operators **in notin exists**
  + Let’s try some examples 

### Annotations

* Using labels we can add metadata which can be later used to filter/select objects
* Annotations on the other hand have fewer constraints, however we cannot filter or select objects by annotations
* Tools or users to get subjective information regarding k8s object generally use annotations.
* Let’s look at one annotation example [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/4af32e39ca4b0f87a234b0273da4e3b035bdc0f5) for the manifest 
* Like what we have done in labels we can add/modify/delete annotations from kubectl

kubectl annotate pod <podname> <annotate\_key>=<annotate\_value>

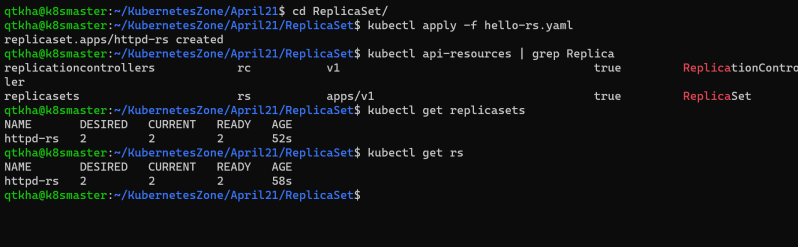
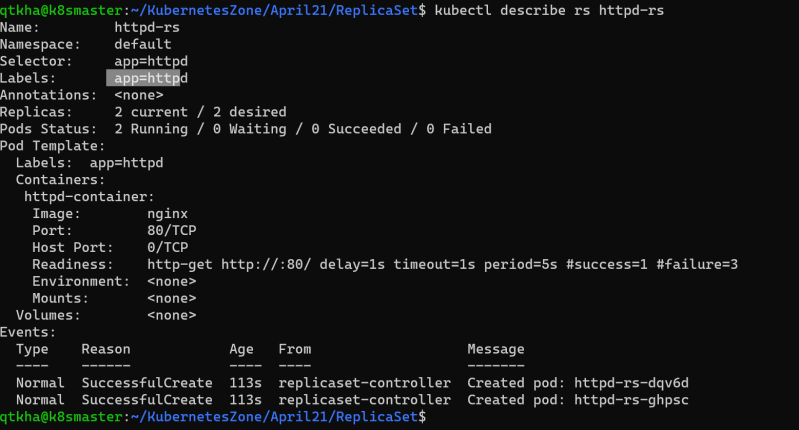
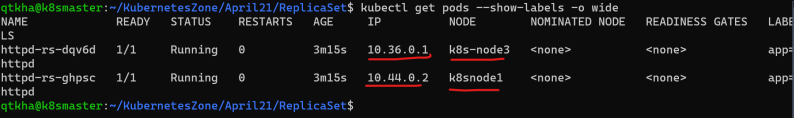
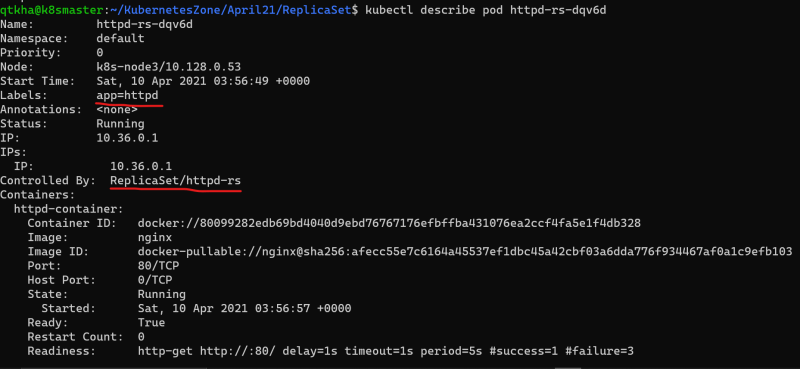
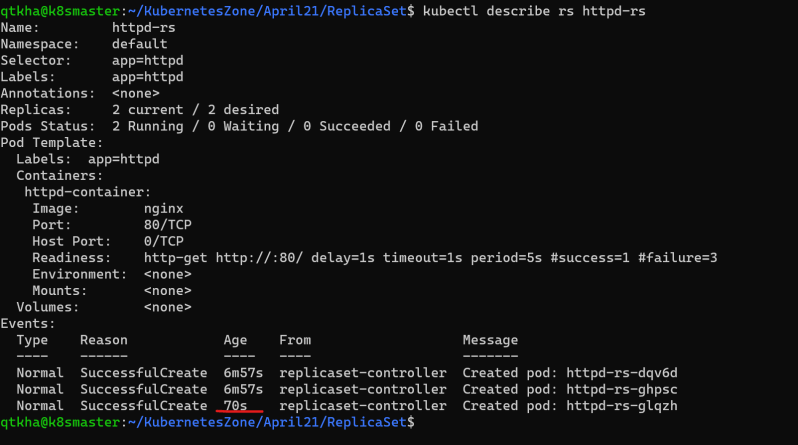
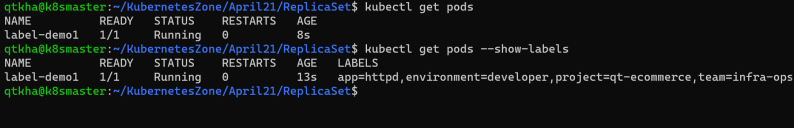
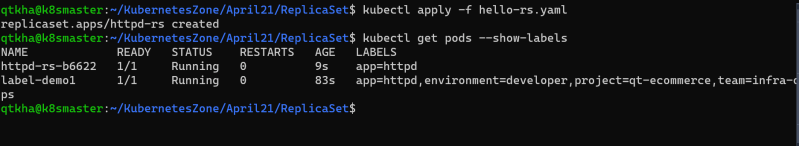
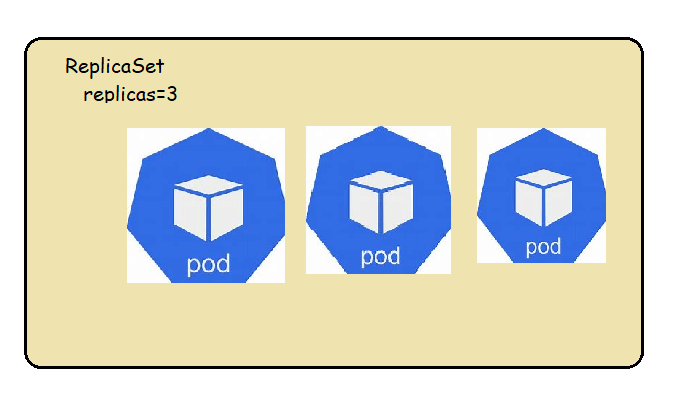
kubectl annotate --overwrite pod <podname> <annotate\_key>=<annotate\_value>

kubectl annotate pod <podname> <annotate\_key>-

### Kubernetes Controllers

* Pod is the workload in the k8s cluster
* When we deploy our application in production,
  + we might need more than one replica of pod
  + We can balance load across multiple pods so that one pod is not overloaded
* k8s supports different controllers that you can use for replications and maintaining state. The controllers we have are
  + Replica Sets
  + Deployments
  + Daemon Sets
  + Stateful Sets
  + Jobs
* A controller is an object in k8s that ensures application runs in the desired state for its entire runtime

#### **Replica Set**

* A Replica Set is a k8s controller that keeps a certain number of Pods running at any given time
* Replica Set acts a supervisor for multiple Pods across different nodes in a k8s cluster. A Replica Set will terminate or start new Pods to match the configuration specified in Replica set Template.
* Create a sample replica set [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/0e1b2278b7d2131efc6b92499268b178d50d34b4) for the yaml and apply    
* Let’s delete one pod manually and see what happens  
* Now lets try to delete the rs 
* Now lets create a pod with app: httpd label [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/ea2976139c1710a2097829efd8e5d7acc907f874) for yaml and apply this 
* Now lets apply the replica set which needs 2 replicas 
* replica set didn’t create 2 pods rather it created one pod as there was one pod which matched the label selctor mentioned in replica set specification 
* Manually Scaling the replicas
  + change the spec
  + executing the command 
* Note: command to create token and display the join command in k8s custer

kubeadm token create --print-join-command --ttl=0

kubeadm token list

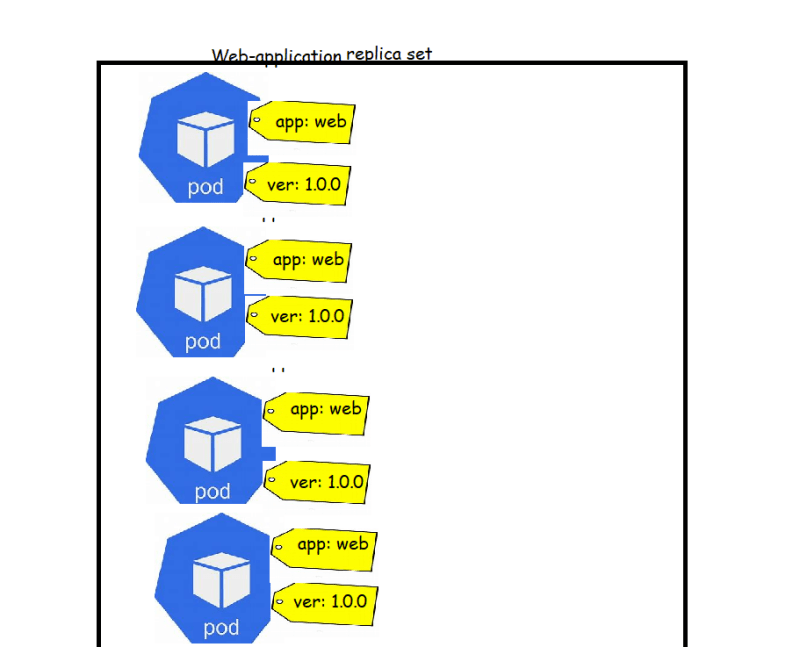
* Exercise: Try to create replica set with 3 replicas
  + for running game of life
  + for running spring pet clinic
  + for nop commerce

### Share this:

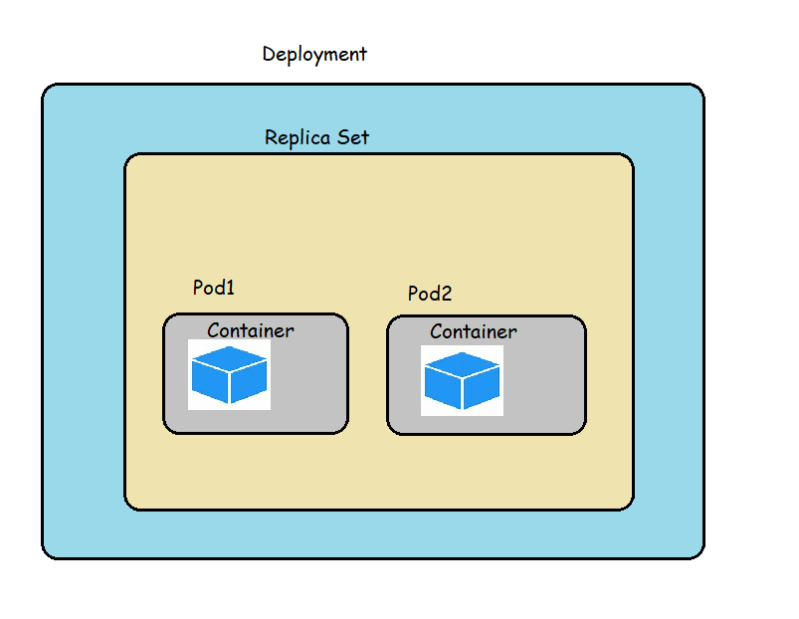
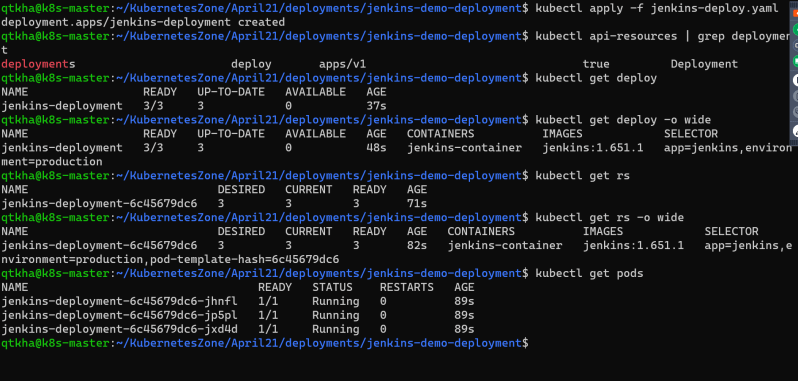
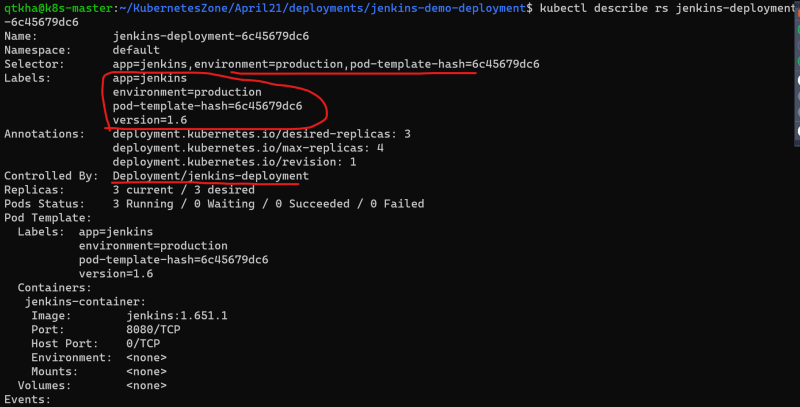
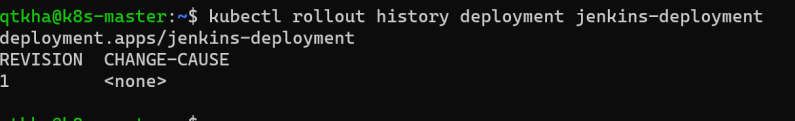
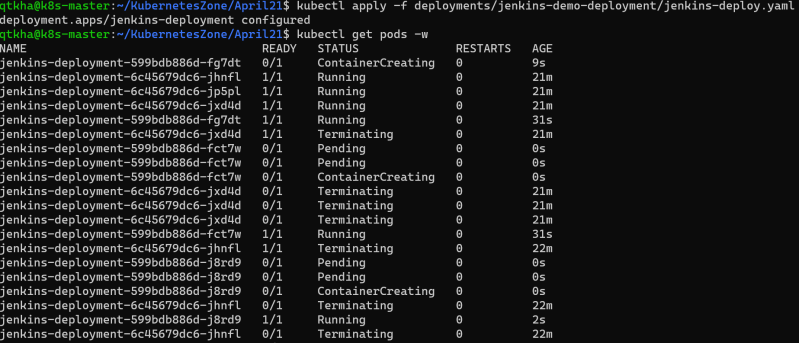
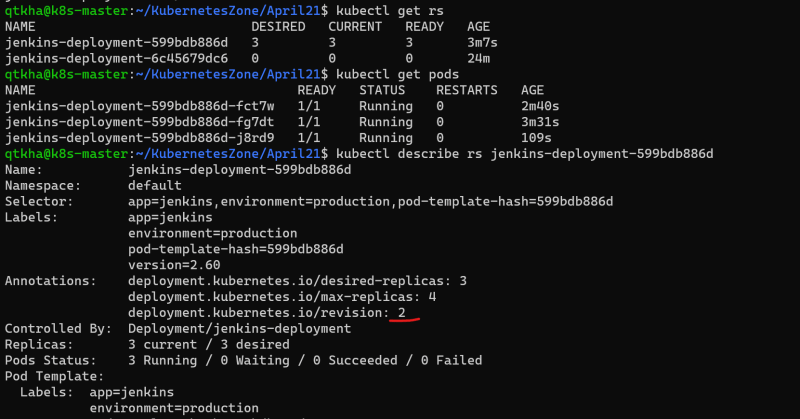
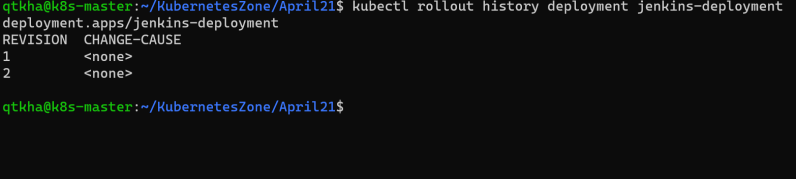
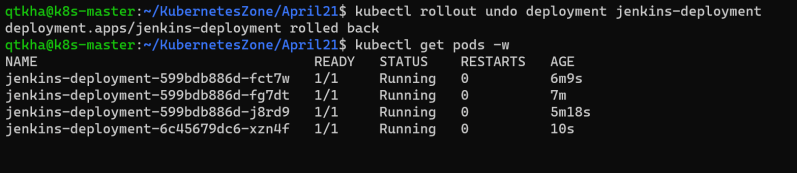
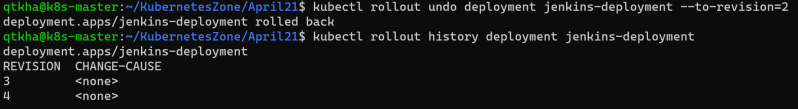
APRIL 10, 2021

# DevOps Classroom Series – 10/Apr/2021

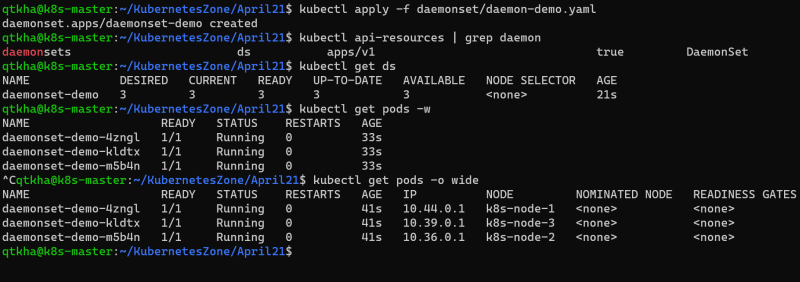
### Introducing the problem

* Consider we have replica set where we deployed the application with 4 replicas 
* Now when we get a new release from dev team, we build a new docker image and upload it to the docker hub or any private registry
* How can we apply the new image to replica set?
  + one approach is to change the replica set and apply new changes (new docker image) but this will have downtime
* We need a solution where we can apply the new docker images with replications without having downtime
* This can be achieved by kubernetes Deployment controller object

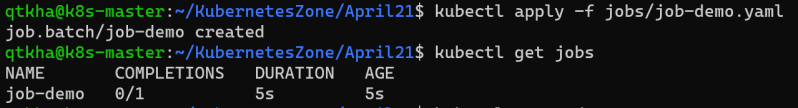
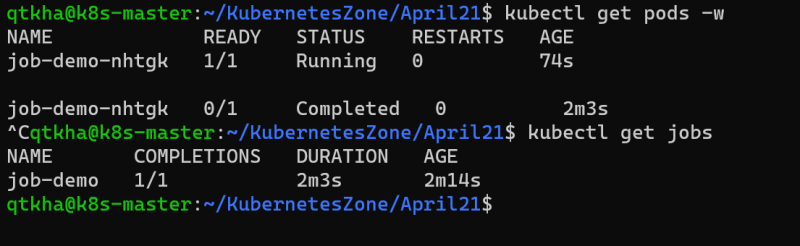
### Deployment

* A deployment is a k8s object that acts a wrapper around Replica Set and makes it easier to use.
* The major motivation for deployments is to maintain the history of revisions.
* Every time a change is made to the replica set or the underlying pods a new version of the Replica Set is recorded by Deployment. This way using a Deployment makes it easy to rollback to previous state or version
* Hierarchy of Deployment, Replica Set, Pods and container 
* Strategy: In this strategy section we can define strategy to replace old pods with new ones when you are updating . There are two kinds of strategies
  + Rolling Update
  + Recreate
* **Rolling Update**:
  + We use this strategy to update deployment without having a downtime
  + In Rolling update we specify two important values
    - Max Unavailable: This is max number of Pods that can be unavailable during the update. default value for max Unavailable is 25%
    - Max Surge: this is number of Pods that can be scheduled/created above the desired number of Pods. default value for max Unavailable is 25%
* **Recreate**: In this strategy all the existing pods will be killed b4 creating new pods, so there will be some down time
* Let’s create a simple deployment
* [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/d384934169deb06a58af4a71a20a1ecdcffd955b) for the change set
* Let’s apply the template  
* Labels and annotations on the deployment
  + k8s adds an annotation with the deployment.kubernetes.io/revision key which contains information about how many revisions have been there for particular deployment 
* Added a simple service to access Jenkins [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/cda6a4ec2021660de6981d07f6457e94f94419ef) for spec
* Now make some changes to simulate new application version [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/0affe7099f08f12d9eb70316c3a1eea74fa026b1) for the changes
* Now let us apply the changes using kubectl apply with –record flag. This flag ensures that the update to the deployment is recorded in the history of deployment   
* Let’s undo the deployment  

### Daemon Sets

* Daemon Sets are used to manage the creation of particular pod on all or set of selected nodes in a cluster
* Use cases for Daemon Set
  + Logging: To manage running a log collection pod on all nodes and then process the logs in a log processing pipeline (sending logs to central logging server)
  + Local data caching
  + Monitoring
* Let’s write one daemon set
  + [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/56fd517b90555b33c05b21e93166a6e88892feae) for the change set
  + Let’s apply the template 

### Jobs

* A job is supervisor in k8s that can be used to manage pods to run some script/task and exit gracefully
* The pods created by the job are not deleted following the completion of job. The pods run to completion and stay with a Completed status
* Let’s create a k8s job configuration which will run the alpine pod to sleep for 2 minutes (sleep 2m) with restart Policy On Failure
* [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/1f37af435f42e3a587cc8c85b92e4c8d30cde288) for the changeset  
* Exercise: Try to create a cron job spec which run every 15 minutes using same pod spec

### Share this:

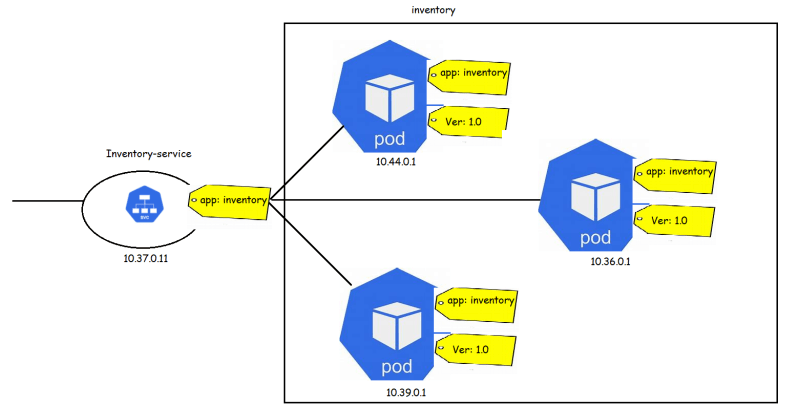
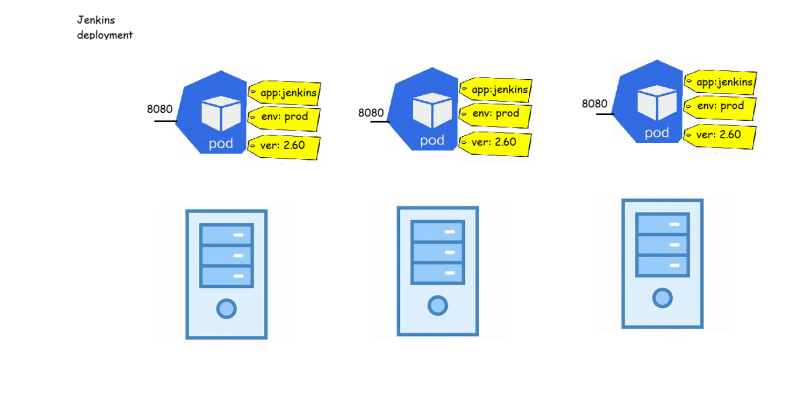
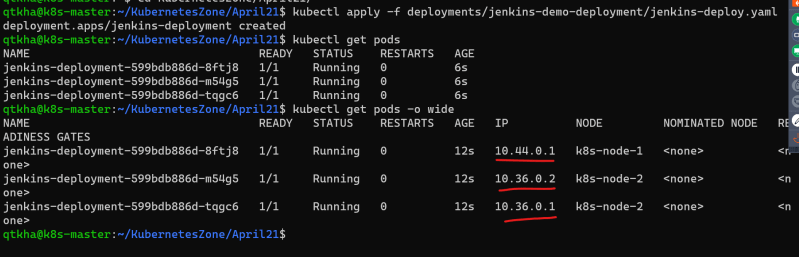
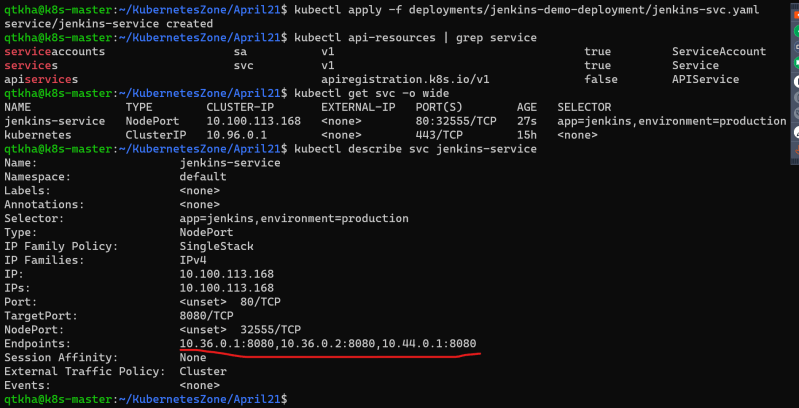
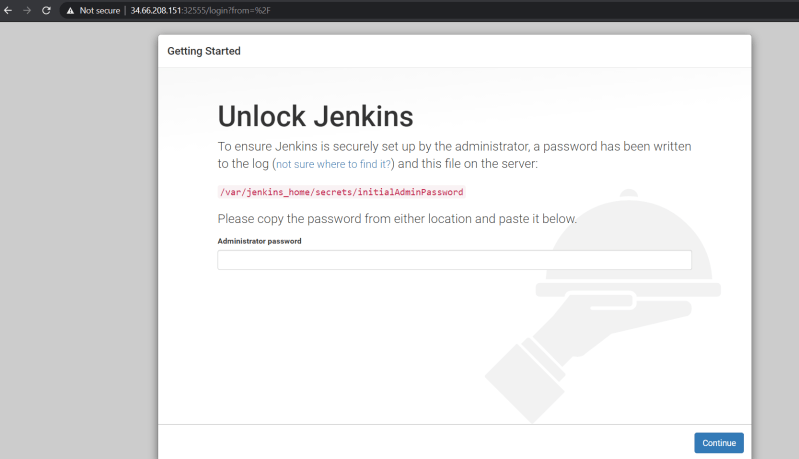
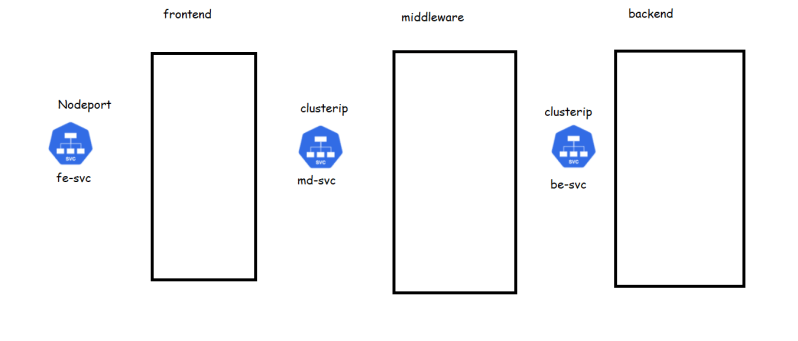
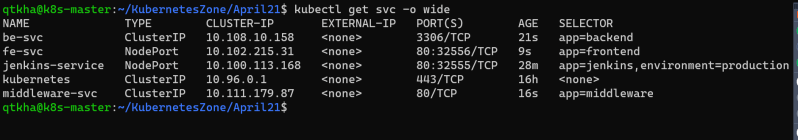
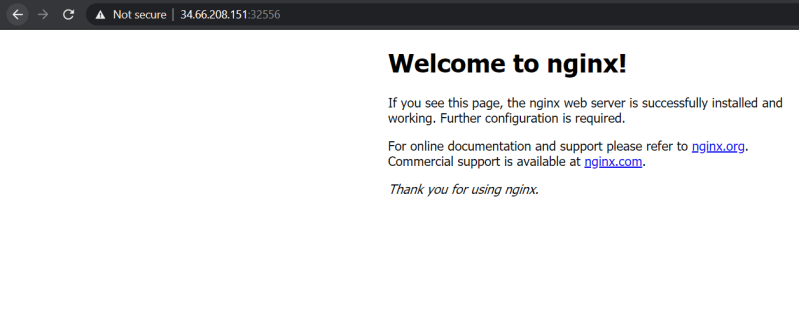
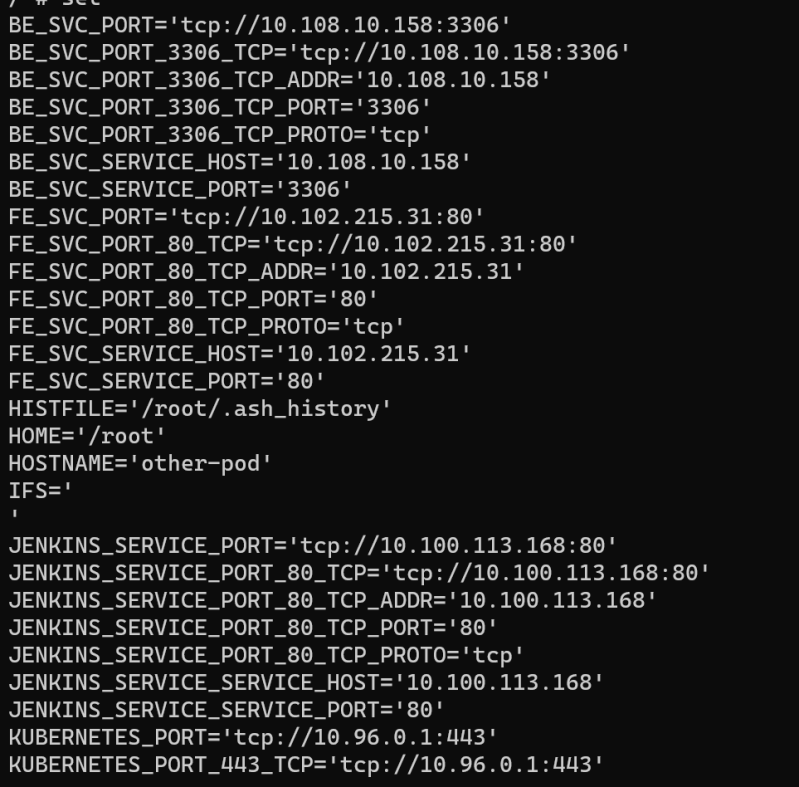
APRIL 11, 2021

# DevOps Classroom Series – 11/Apr/2021

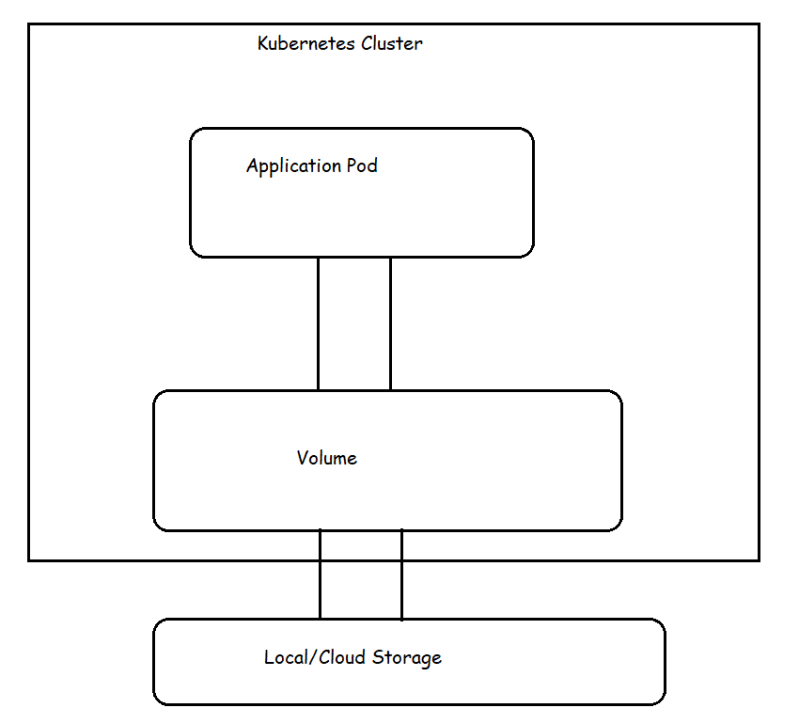
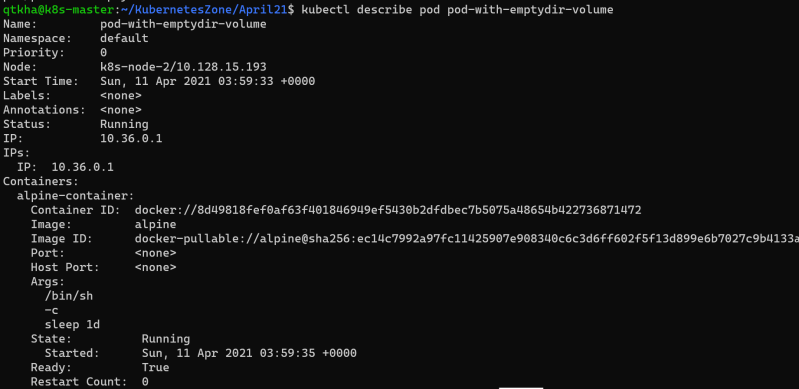
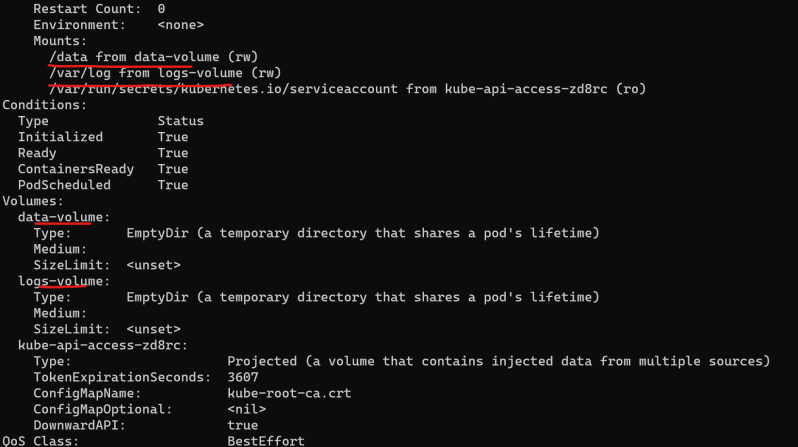
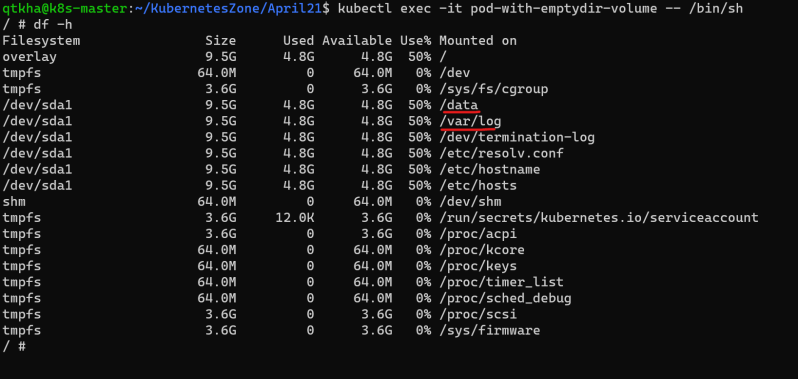
### Service Discovery

* So far we have learnt how to create Pods, Deployments and other controllers which help in deploying the containerized applications.
* We also have learnt each pods gets IP address.
* How can we connect to the applications running in the pods from external and also with in cluster.
* To help with this we have Kubernetes services. Services allows us to make logical set of pods discoverable and accessible for other pods running inside the cluster or to the external world

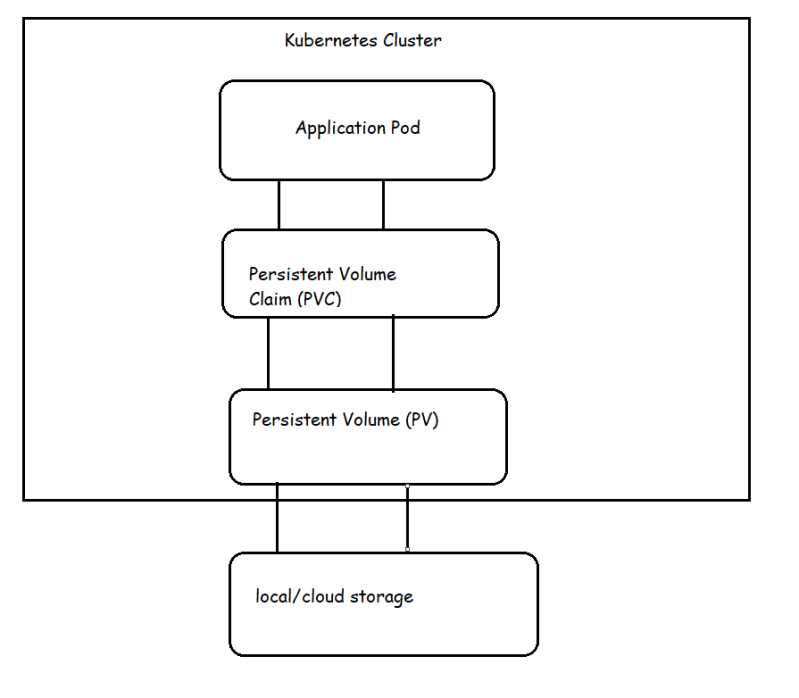
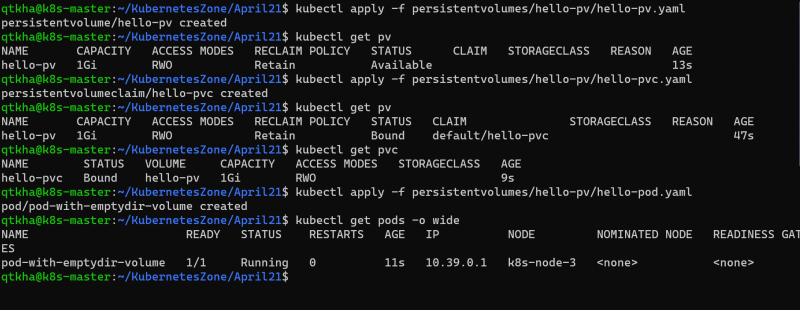
#### **Kubernetes Services**

* Basic workflow 
* Types of Services
  + Node Port: This type of services makes internal pods accessible on the port of the node on which pods are running
  + Cluster IP: This type of service exposes the Service on a certain Ip address with in cluster
  + Load Balancer: This type of Services exposes application externally using the load balancer provided by cloud provider
  + External Name: This type of service points to a DNS rather than set of pods
* Let’s try to use the Jenkins deployment [Refer Here](https://github.com/asquarezone/KubernetesZone/tree/master/April21/deployments/jenkins-demo-deployment)
* Let’s create a Jenkins deployment [Refer Here](https://github.com/asquarezone/KubernetesZone/blob/master/April21/deployments/jenkins-demo-deployment/jenkins-deploy.yaml)
* The deployment can be represented as shown below  
* Now lets try to create a Jenkins service which exposes this application on some port of the node
* [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/82eba35c330c4748b52c7030cbdca3c54dbbd56d) for the Jenkins-service created
* Create a Jenkins service  
* Let’s create one simple test pod to check certain things [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/577a2ab3d18106af1c804578e42ff559f2484efe) for the template and create the pod
* Now login into container and execute
  + curl [http://ipaddressofsvc](http://ipaddressofsvc/)
  + set to see environment variables
* Now let’s create a deployment
  + of nginx containers and call it as frontend-deployment
  + of http containers and call it as middleware-deployment
  + of mysql container and call it as backend 
* Now let’s add three service templates [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/e535f350c35fe53f615bae773891722e2d492eaf) for the changes
* apply the templates  
* Login into verification pod and try to access the mw-svc 
* Load balancer and ingress samples will be done with cloud providers
* [Refer Here](https://www.digitalocean.com/community/tutorials/an-introduction-to-haproxy-and-load-balancing-concepts) to understand differences layer 4 and layer 7 load balancing

#### **Kubernetes Volumes**

* If the containers that store the data crashes and is restarted the data will be lost. The new container will start with empty disk space allocated
* Volume as a storage abstraction for pod applications 
* Lifetime of k8s volume is same as pod that uses it.
* Even if the containers in the pod restart, new container will use the same volume as well. Data will not be lost across container restarts, but when the pod is terminated or restart the volume ceases to exist (volume will be deleted)
* If you want to solve this problem of volumes getting deleted when pods get terminated then we need to use persistent volumes
* Types of volumes
  + Emptydir:
  + hostPath: this is used to mount a file or directory from the host node’s file system to a pod
  + There are other cloud based volumes [Refer Here](https://kubernetes.io/docs/concepts/storage/volumes/#volume-types)
* Let’s try a sample pod with volume
  + [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/d7df786fdc333365950bbe6f8fdd1ada23f36259) for the change set
  + Let’s create the pod  
  + Let’s log in into this pod 
  + Multiple containers sharing volumes
* [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/a19d716a7d44bae01af061a2643326535f90d18d) for the host Path demo

### Persistent Volumes

* Kuberentes supports persistent storage in the form of Persistent Volume (PV).
* PV is a kubernetes object that represents block of storage in the cluster.
* This can be either be provisioned beforehand by the admins or we can dynamically provision
* A PV is a cluster resource. Lifecycle of PV doesn’t depend on lifecycle of pod.
* In order to use a PV, a Persistent Volume Claim (PVC) needs to be created.
* A PVC is a request to the storage by the user or pod.
* PVC can request specific size of storage and specific access mode
* PVC are scoped by namespace
* How PV and PVC work together to provide the storage to application pod 
* k8s supports different types of Persistent Volumes [Refer Here](https://kubernetes.io/docs/concepts/storage/persistent-volumes/#types-of-persistent-volumes)
* Each PV belongs to certain **storage class**.
* Persistent Volume Reclaim policy
  + Retain
  + Recycle
  + Delete
* PV Status
  + Available
  + Bound
  + Released
  + Failed
* Access Modes [Refer Here](https://kubernetes.io/docs/concepts/storage/persistent-volumes/#access-modes)
* [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/3f63af073f5a72004801d29178865fe85f3bf24e) for the sample configuration 

### Next topics

* Config Map and Secrets
* Azure Kubernetes Services
* Elastic Kubernetes Services
* Google Compute Engine

#### **Topics for next weekend**

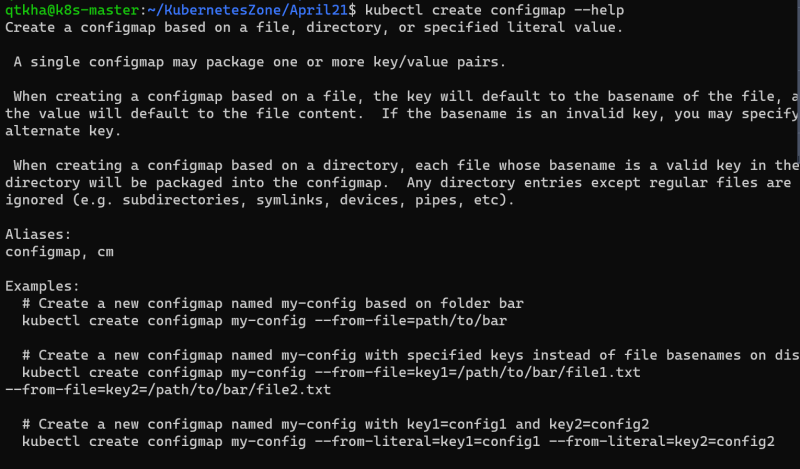
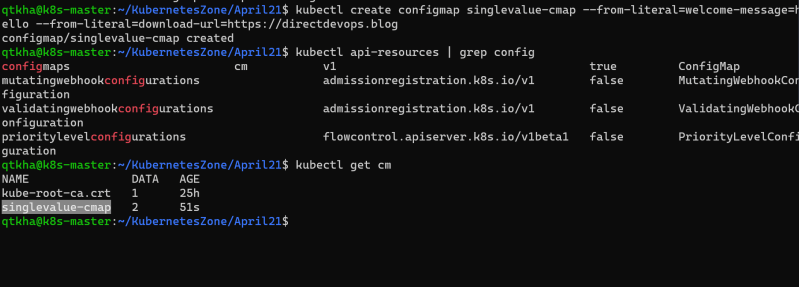
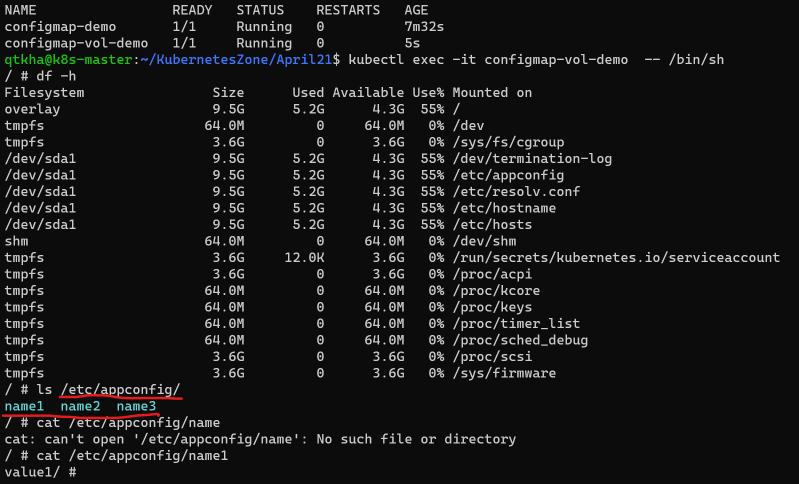
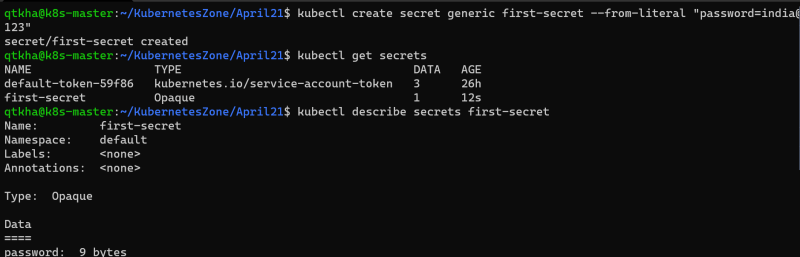
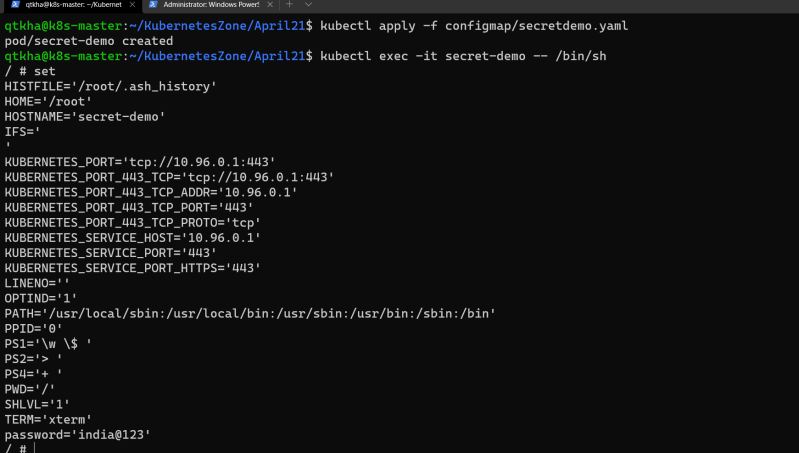
* Helm Charts
* Openshift
* Istio

### Share this:

APRIL 11, 2021

# DevOps Classroom Series – 11/Apr/2021

### Config Maps & Secrets

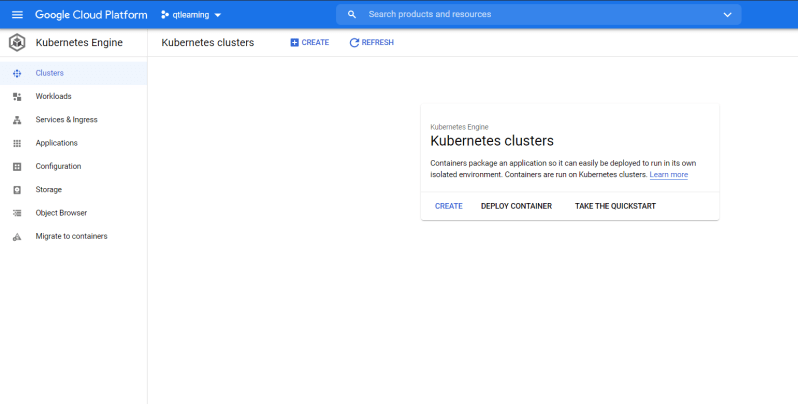
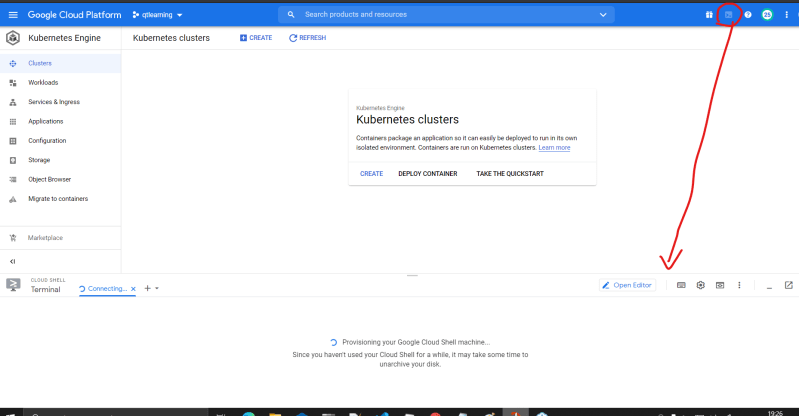
* A ConfigMap allows us to define application related data.
* ConfigMap can be create through a literal value or from a file 
* Let’s create a config map from literal values and mount it to the pod Preview 
* Now let’s create a pod which mounts the config map [cx](https://github.com/asquarezone/KubernetesZone/commit/624058c9fb52fd1e7ab0f22d9d2f68c83a3bf6d7) for the changes and create the pod
* Config maps can be mounted to the pods as volumes as well [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/2d9143e7e23b7e5d2c2bcdbb99b3c316041ac56c) 
* Secret is also much like config map but in secrets the values are base64 encoded
* Kubernetes secretes has 3 available commands
  + generic: generic secret holds any key value pair
  + tls: secret for holding private-public key for communicating with TLS protocol
  + docker-registry: This is special kind of secret that stores usernames and passwords to connect to private registries
* Create a secret  
* Like configmap secrets also can be mounted as a volume

#### **Kuberenets as a Service on Cloud Platforms**

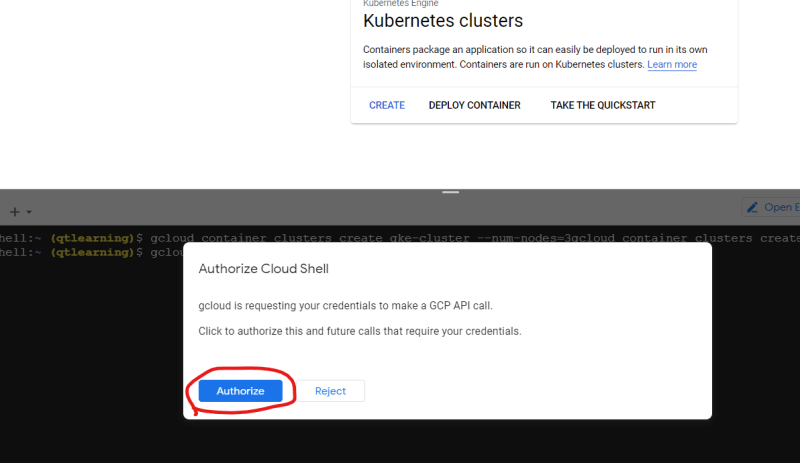
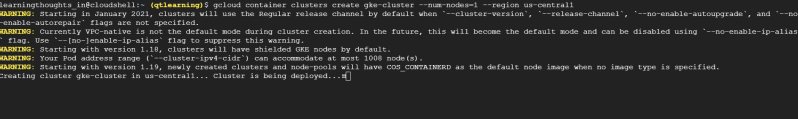
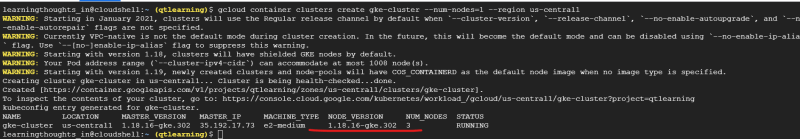
* Cloud providers like AWS, Azure , Google offer kubernetes as a service
* When we use these offerings
  + Google Kuberenetes Engine
  + Azure Kubernetes Services
  + AWS Elastic Kubernetes Services
* The cloud provider will manage
  + the k8s master nodes
  + the networking configuration
  + Load balancing and ingress capabilities
  + Persistent Volume native support to the clouds block and file storage
  + Integrated logging and monitoring support

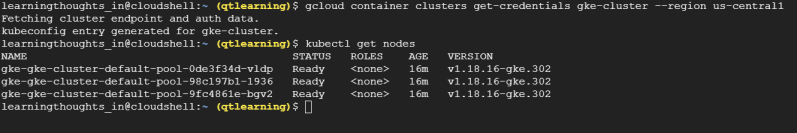
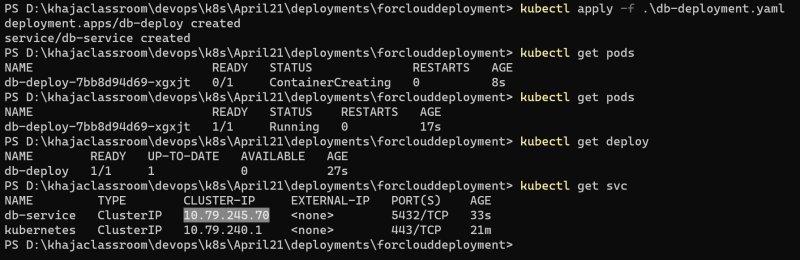
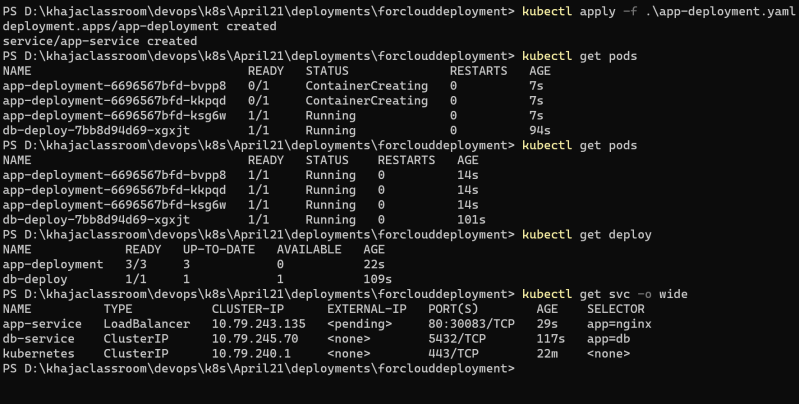
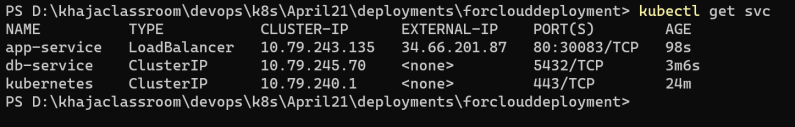
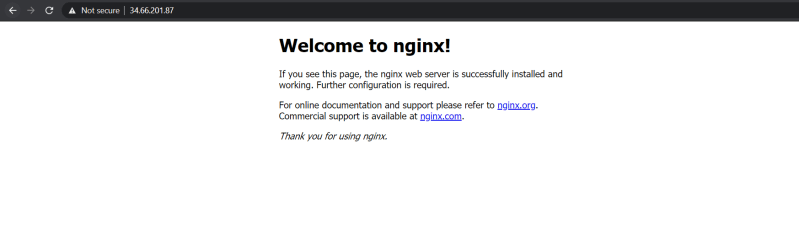
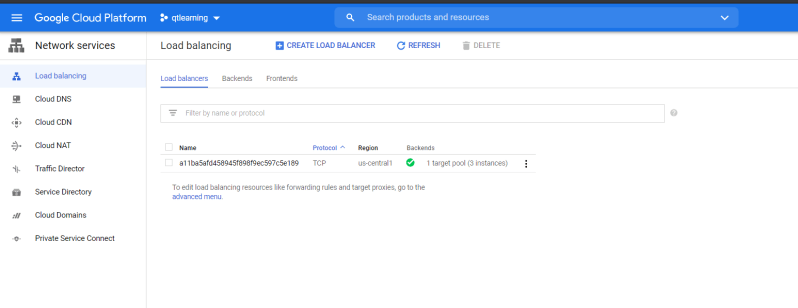
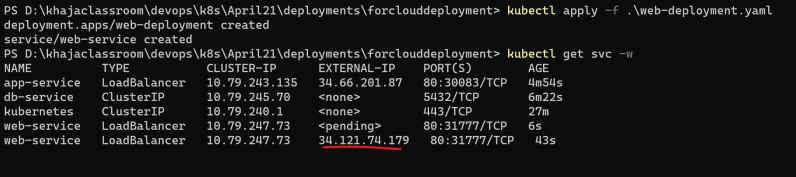
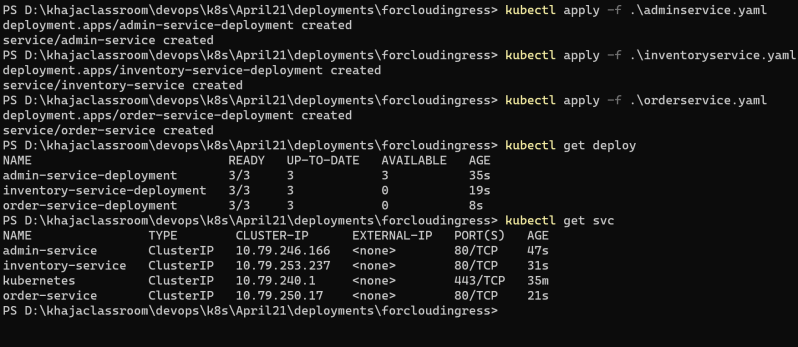
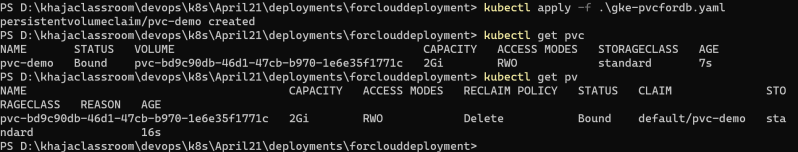
break till 7:25 PM

### Google Kubernetes Engine (GKE)

* GKE is fully managed k8s cluster offered by GCP  
* Let’s create a k8s cluster [Refer Here](https://cloud.google.com/sdk/gcloud/reference/container/clusters/create) and also navigate to the other sections
* Create a 3 node cluster

cloud container clusters create gke-cluster --num-nodes=1 --region=us-central1

* Now to configure the kubectl execute the get-credentials [Refer Here](https://cloud.google.com/sdk/gcloud/reference/container/clusters/get-credentials) 
* Now let’s try to use the deployments and services created [Refer Here](https://github.com/asquarezone/KubernetesZone/tree/master/April21/deployments/forclouddeployment)  
* Let’s wait till load balancer get the external ip and access the application using the load balancer external ip  
* k8s has create a load balancer in GCP
* Now let’s try to apply the web deployment 
* Let’s try to create ingress load balancing using the examples [Refer Here](https://github.com/asquarezone/KubernetesZone/tree/master/April21/deployments/forcloudingress) 
* For gcp lets apply the annotation as specified
* Now lets try to create the persitent volume dynamically and use it for postgres volume [Refer Here](https://cloud.google.com/kubernetes-engine/docs/concepts/persistent-volumes#dynamic_provisioning)
* 
* To be shared
  + manifest with correct annotations to run the ingress controller
  + manifest to use postgres with gke persistent volumes
* [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/ccdd1da7ac51034de5a92b0d89fe7ded332fd7a1) for the pvc sample
* Note: [Refer Here](https://directdevops.blog/2019/11/02/deploying-the-docker-application-and-mysql-with-volume-support-into-kubernetes-from-code-to-docker-registries-like-acr-ecr-and-then-to-eks-aks/) to understand the flow of the application creating the docker file and running it on k8s cluster.

### Next Weekend topics

* AKS
* EKS
* HELM CHARTS
* ISTIO
* Open shift

### Share this:

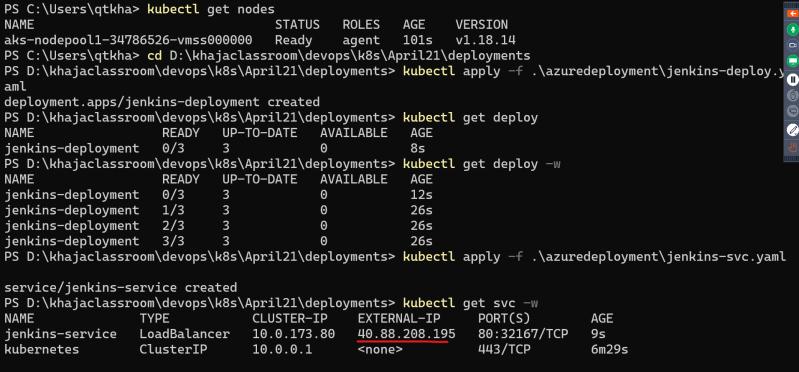
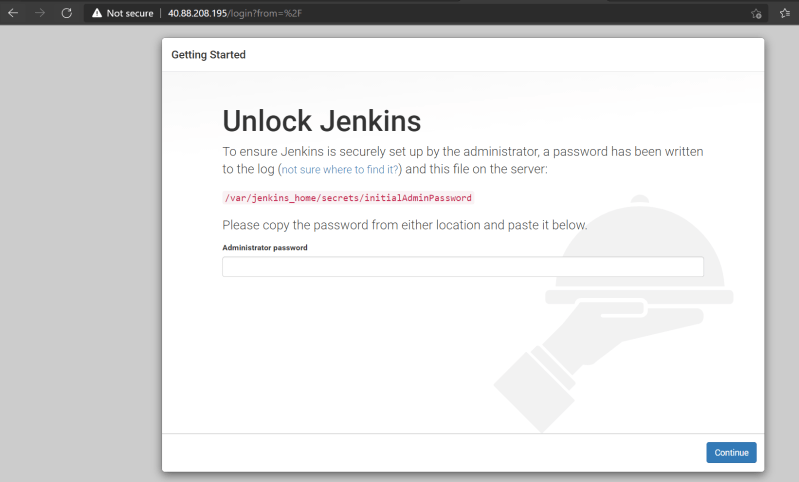
APRIL 17, 2021

# DevOps Classroom Series – 17/Apr/2021

### GKE ingress issue

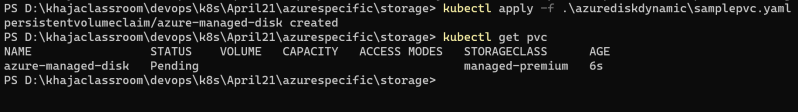
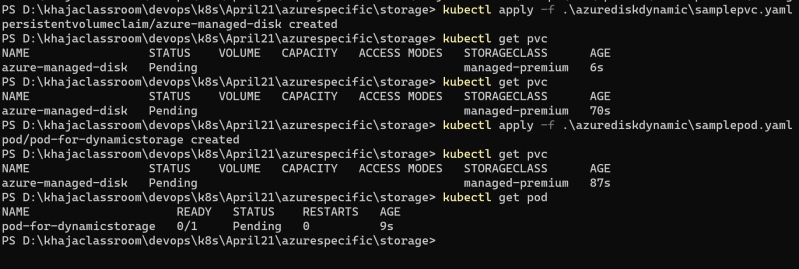
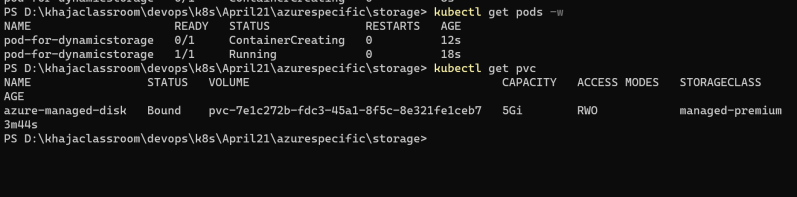
* IN GKE when we use ingress the service type should be Node Port or Load balancer
* the fix is submitted over here [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/f9439b48eed88269ec305ea3d1b0cbbffb39f82e)

### Azure Kuberenetes Services (AKS)

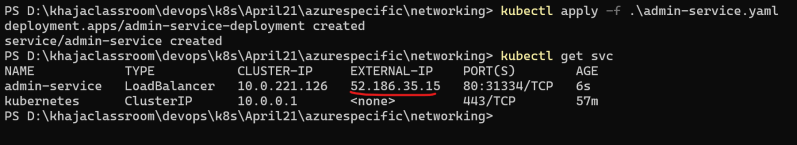
* AKS simplifies deploying a managed k8s cluster by offloading the operational overhead to Azure
* Azure handles
  + health monitoring
  + maintenance
* Azure doesn’t charge for cluster
* We can create an AKS Cluster from Azure CLI/Azure Power shell/Azure Portal
  + Azure CLI [Refer Here](https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough)
  + Azure Power shell [Refer Here](https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough-powershell)
  + Azure Portal [Refer Here](https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough-portal)
* Let’s deploy a sample deployment in aks  
* Now delete the deployment
* Scaling number of nodes in aks cluster

az aks scale --resource-group myResourceGroup --name myAKSCluster --node-count 3 --nodepool-name nodepool1

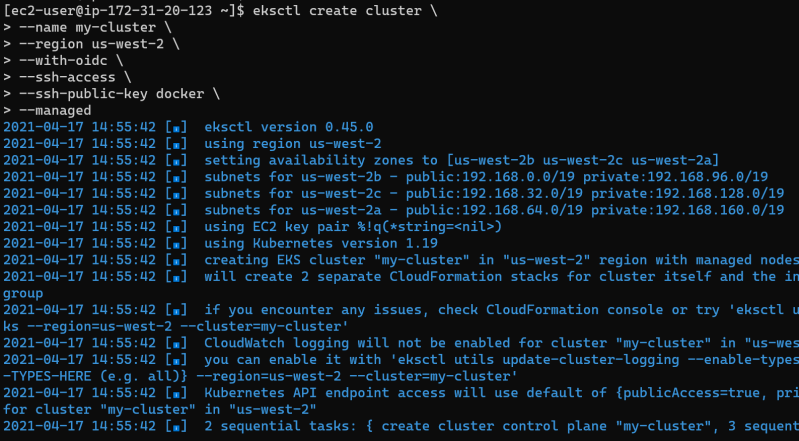
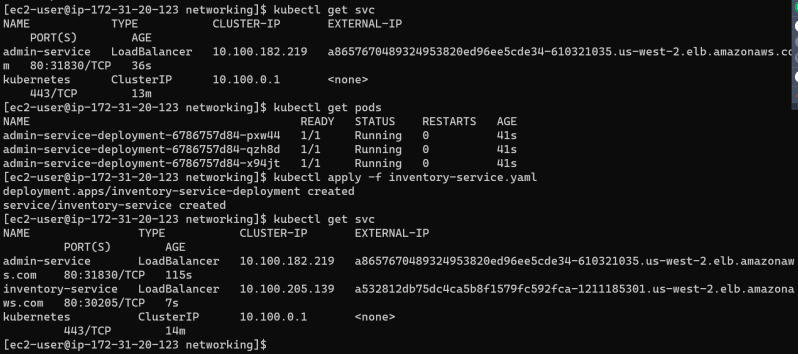
#### **Storage in AKS**

* To define different tiers of storage such as premium and standard we can define storage classes.
* In AKS we have four initial Storage Class
  + Default: Uses Azure Standard SSD storage to create a managed disk.
  + managed-premium: Use azure Premium storage to create managed disk
  + azure file: uses standard storage to create an Azure file share
  + azure file-premium: uses premium storage to create an Azure file share
* Let’s create a sample pod to use Azure Disk with dynamic provisioning [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/5bbd32c05cfcd3d0eb1e068233c17e8a734ddc43)   
* [Refer Here](https://docs.microsoft.com/en-us/azure/aks/azure-disk-volume) to create a static disk
* [Refer Here](https://docs.microsoft.com/en-us/azure/aks/azure-files-dynamic-pv) to create a dynamic pv using azure file

#### **Network in AKS**

* Let’s use a load balancer in Azure [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/53adb3b99b5480f230f4720eb50dc6a4e8f466db) for the change set
* Deploy the service and deployment  
* [Refer Here](https://docs.microsoft.com/en-us/azure/aks/http-application-routing) for creating an ingress
* Sample used [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/1b7e3036043a9203fe52d7b24d7cc8411ddce4bf) 

### AWS Elastic Kubernetes Services (EKS)

* AWS EKS is a managed service that we can use to run k8s on AWS without needing to install, operated and maintain own k8s control plane
* Install eksctl, kubectl and ensure iam is configured
* Now create the cluster following the documentation [Refer Here](https://docs.aws.amazon.com/eks/latest/userguide/getting-started-eksctl.html) 
* [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/5a2cc7705b09550cb3c7f3183da6cef4ba26c32b) for the load balancing sample
* Let’s create basic networking with load balancers 
* To create ingress [Refer Here](https://docs.aws.amazon.com/eks/latest/userguide/alb-ingress.html)
* annotations in aws for load balancers [Refer Here](https://kubernetes-sigs.github.io/aws-load-balancer-controller/guide/ingress/annotations/)
* [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/3eb3b8aa34d479369da631a2d4bdeb2661f546b4) for the ingress sample
* [Refer Here](https://directdevops.blog/2019/11/02/deploying-the-docker-application-and-mysql-with-volume-support-into-kubernetes-from-code-to-docker-registries-like-acr-ecr-and-then-to-eks-aks/) to understand storage and networking in aws and other clouds

### Next Steps:

* Horizontal pod auto scaling
* docker private registries
* Helm Charts
* Istio Service Mesh
* Open shift

### Share this:

APRIL 19, 2021

# DevOps Classroom Series – 18/Apr/2021

### Horizonal pod auto scaler HPA

* HPA automatically scales number of pods in Replica Set, deployment based on observed CPU utilization or with custom metrics support.
* Deploy any application and execute the following command

kubectl autoscale deployment admin-service-deployment --cpu-percent=50 --min=1 --max=5

* [Refer Here](https://github.com/asquarezone/KubernetesZone/commit/4461127f04f7913cfc8c24ee399dfbeef9841899) for the sample hpa
* [Refer Here](https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/#support-for-resource-metrics) for official docs

### Share this:

APRIL 19, 2021

# DevOps Classroom Series -18/Apr/2021

### Package management

* Package managers are used to simplify the process of installing, upgrading, reverting and removing system’s applications.
* The application is defined as package, which contains metadata around target software and its dependencies

dnf install htop

dnf upgrade htop

dnf downgrade htop

dnf remove htop

## Helm: The Kubernetes Package manager

* In the above-mentioned examples dnf operates on RPM package and apt operates on deb packages that provide executables, dependency information, and metadata.
* Helm works with charts. This chart can be thought of k8s package.
* Charts contain declarative k8s resource files that are required to deploy an applications
* Example: let’s assume you want to deploy Redis as memory cache in k8s, using helm we can perform this

# install

helm install redis bitnami/redis --namespace=redis

# upgrade

helm upgrade redis bitnami/redis --namespace=redis

# downgrade

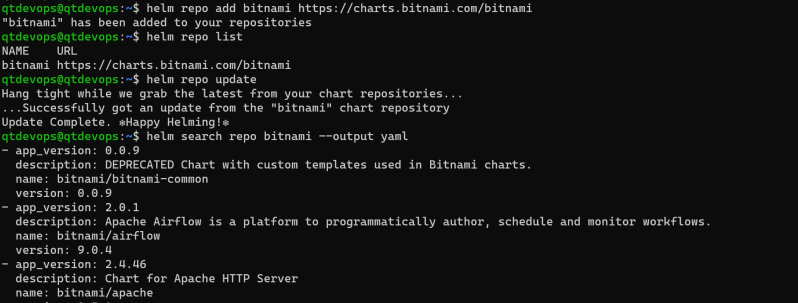
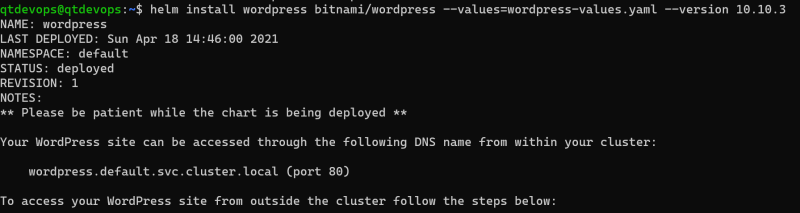
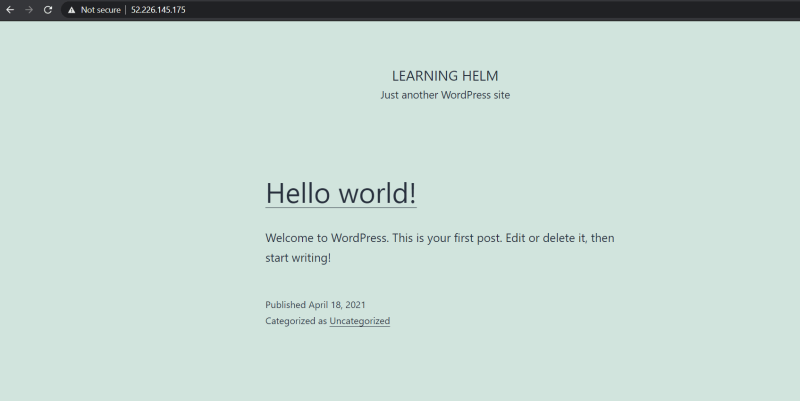
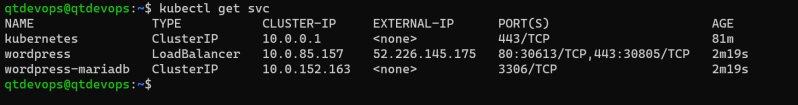
helm rollback redis 1 --namespace=redis

# uninstall

helm uninstall redis --namespace=redis

* Lets search for WordPress

helm search hub WordPress

* add wordpress repository 
* Lets install wordpress chart   
* Using helm charts is extremely simple and convinient

### Creating Helm Chart

* Helm chart has the following files

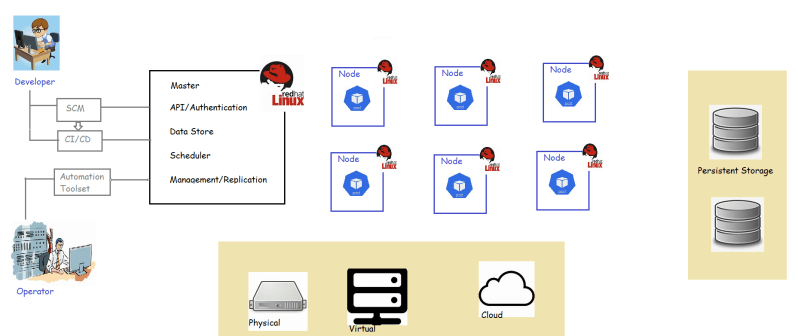
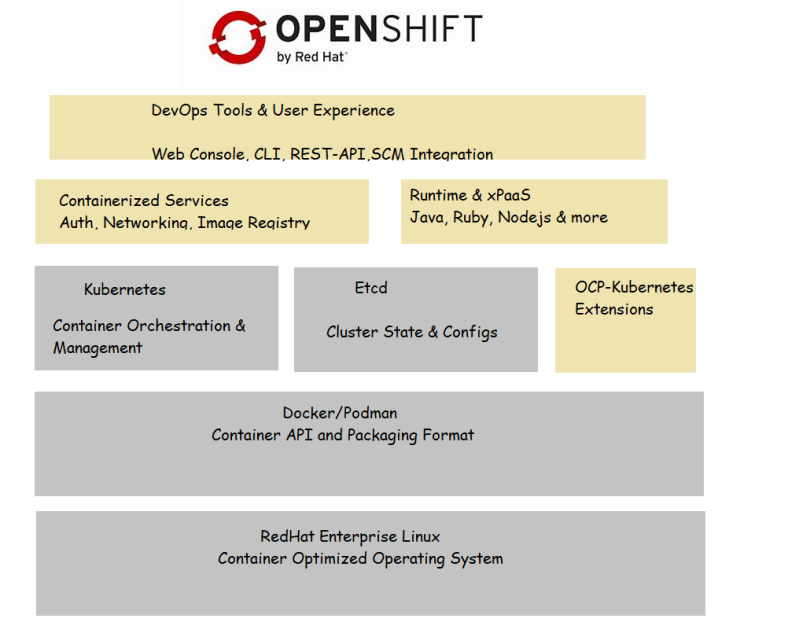
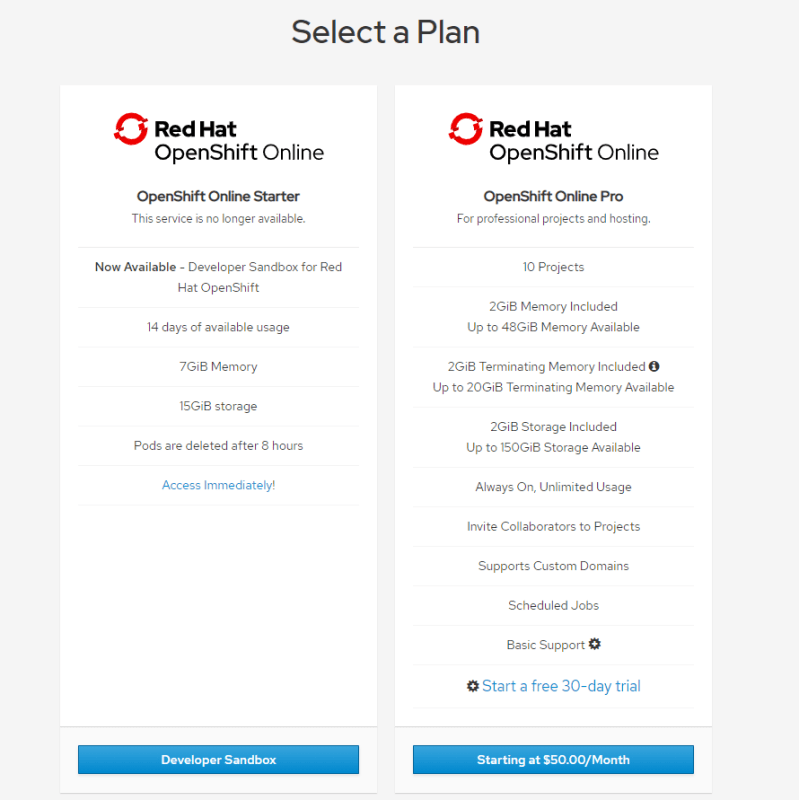
| **File/Directory** | **Definition** | **Required?** |
| --- | --- | --- |
| Chart.yaml | A file that contains metadata about the Helm chart | yes |
| templates/ | A directory that contains k8s resources in YAML | yes |
| templates/NOTES.txt | A file that can be generated to provide usage instructions | no |
| values.yaml | A file that contains charts default values | no |
| .helmignore | A file that contains a list of files or directories to be omitted from Helm charts packaging | no |
| charts/ | A directory that contains charts that the helm chart depends on | Does no need to explicity provided |
| Chart.lock | A file used to see previously applied dependency version | no |

* [Refer Here](https://helm.sh/docs/chart_template_guide/) to understand helm templates
* helm chart YAML [Refer Here](https://helm.sh/docs/topics/charts/#:~:text=yaml%20is%20used%20by%20many,version%20number%20in%20the%20Chart.)
* [Refer Here](https://github.com/asquarezone/KubernetesZone/tree/master/April21/helm/jenkins) for the sample created.

APRIL 19, 2021

# DevOps Classroom Series – 19/Apr/2021

### Open shift

* Open shift is an extension to k8s cluster (Kuberetes ++)
* Open shift used k8s as container management platform & add several capabilities
  + Self-Service portal
  + Service Catalog
  + Build & application deployment
  + Built-in registry
  + Extended application routing
* Open shift Flavors
  + Open shift Origin
  + Open shift Container Platform
  + Open shift Online
  + Open shift dedicated
* Openshift Architecture 
* Openshift Components
  + Openshift masters
  + Openshift nodes
  + etcd as a key value store
  + Router as ingress traffic control
  + Openshift internal registry 
* Openshift developer env can be installed on machines using minishift [Refer Here](https://www.okd.io/minishift/)
* Lets create an openshift developer sandbox at openshift online [Refer Here](https://manage.openshift.com/) 

### Share this:

SEPTEMBER 19, 2021

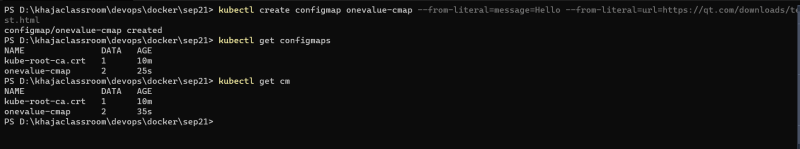
# DevOps Classroom Series – 18/Sept/2021

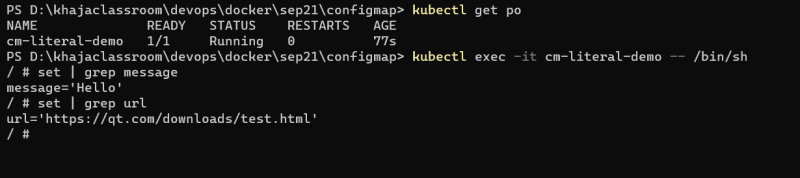
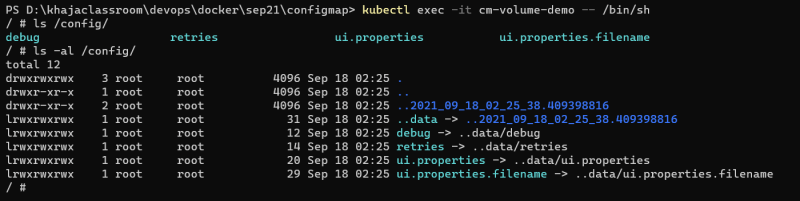
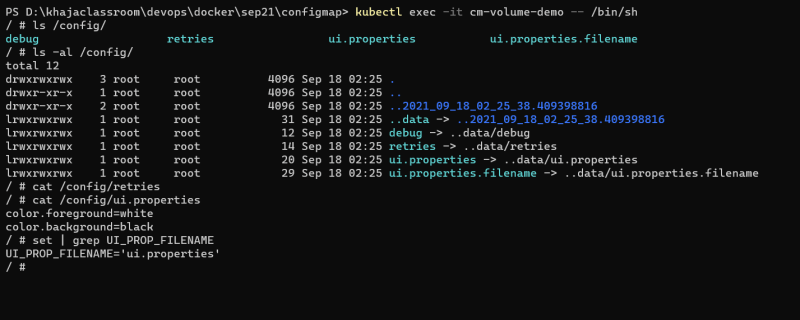
## Config Maps

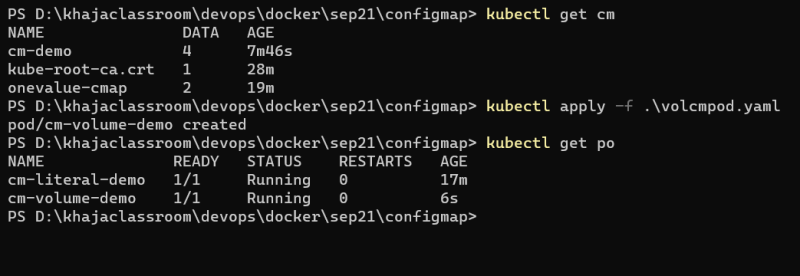
* One way to think of ConfigMap is as a k8s object that defines a small file system.
* Another way is a set of variables that can be used when defining the environment or command line for your container
* In config map we generally define application related configuration data. We can create ConfigMap through a literal value or from a file
* Creating config map from literal values and mount it to the Pod

kubectl create configmap --help

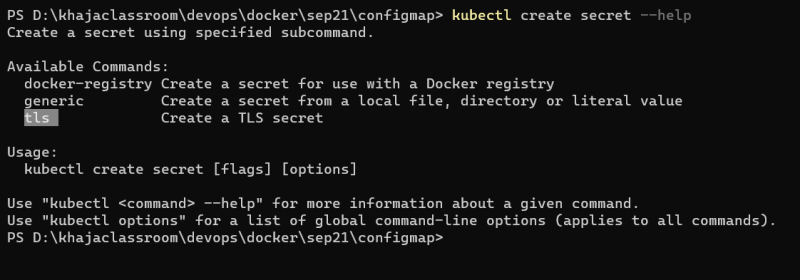
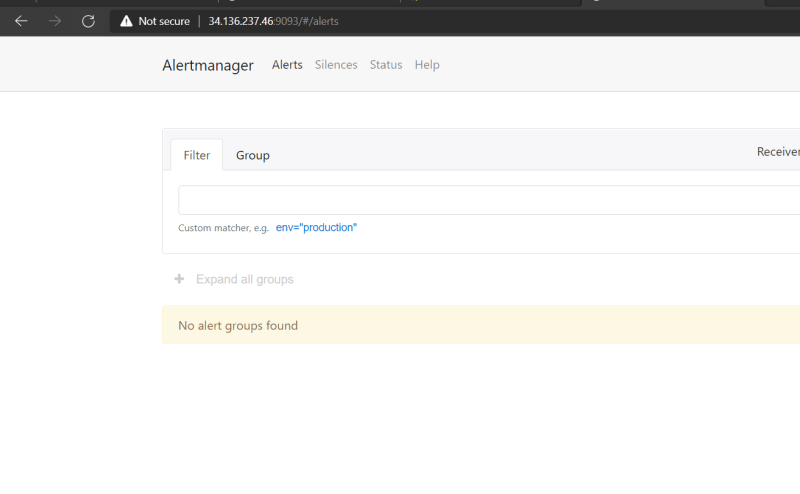
kubectl create configmap onevalue-cmap --from-literal=message=Hello --from-literal=url=https://qt.com/downloads/test.html

* Now lets create a Pod and mount the config maps as environment variables [Refer Here](https://github.com/asquarezone/DockerZone/commit/064895752159014f104e51279048a62d1c2d0a6d) for the changeset and apply this manifest
* Now lets login into the container and check environment variables 
* Now lets create a configmap as Manifest [Refer Here](https://github.com/asquarezone/DockerZone/commit/88d043b981cffcdab2563532bf64671686ce942c) for the changeset and apply the manifests  
* When you change the config maps and want to apply it your deployment use kubectl rollout restart [Refer Here](https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#-em-restart-em-)



## Secrets

* They are much like config map, but in secrets values are base64 encoded  
* Lets create the following secrets

kubectl create secret generic my-secret --from-literal=username=qtdevops --from-literal=password=qtdevops

* [Refer Here](https://github.com/asquarezone/DockerZone/commit/0eb18ccd8c360b5473e8121ddc5814457ccf9449) for the changeset

## Stateful Sets

* Kubernetes statefulsets are useful for running things in cluster e.g hadoop cluset, mysql cluster where each Pod is expected to have its own storage
* In Statefulset also we would have replicas, but each replica will have its own Persitent volume claim placed

apiVersion: v1

kind: Service

metadata:

name: nginx

labels:

app: nginx

spec:

ports:

- port: 80

name: web

clusterIP: None

selector:

app: nginx

---

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: web

spec:

selector:

matchLabels:

app: nginx # has to match .spec.template.metadata.labels

serviceName: "nginx"

replicas: 3 # by default is 1

template:

metadata:

labels:

app: nginx # has to match .spec.selector.matchLabels

spec:

terminationGracePeriodSeconds: 10

containers:

- name: nginx

image: k8s.gcr.io/nginx-slim:0.8

ports:

- containerPort: 80

name: web

volumeMounts:

- name: www

mountPath: /usr/share/nginx/html

volumeClaimTemplates:

- metadata:

name: www

spec:

accessModes: [ "ReadWriteOnce" ]

storageClassName: "my-storage-class"

resources:

requests:

storage: 1Gi

## Elastic Kubernetes Services (EKS)

* To setup the EKS [Refer Here](https://docs.aws.amazon.com/eks/latest/userguide/create-cluster.html)
* In this case i would be using the eksctl way of bringing up kubernetes cluster
* To install eksctl [Refer Here](https://docs.aws.amazon.com/eks/latest/userguide/eksctl.html)
* [Refer Here](https://serverless-stack.com/chapters/create-an-iam-user.html) to create and configure iam user
* Now lets create a k8s cluster

eksctl create cluster --name my-cluster --version 1.21 --with-oidc --without-nodegroup



* Lets add nodes to the eks cluster

eksctl create nodegroup `

--cluster my-cluster `

--region us-west-2 `

--name my-mng `

--node-type t2.large `

--nodes 2 `

--nodes-min 1 `

--nodes-max 2 `

--ssh-access `

--ssh-public-key docker

* Deleting the node group

eksctl delete nodegroup --cluster my-cluster --name my-mng

* Deleting the cluster

eksctl delete cluster --name my-cluster

## Exercise

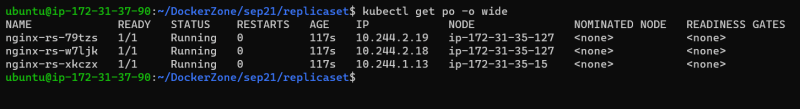
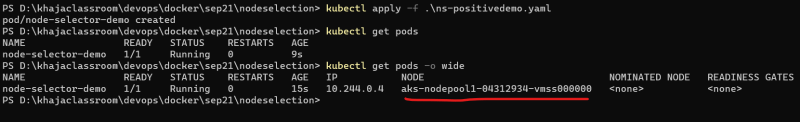
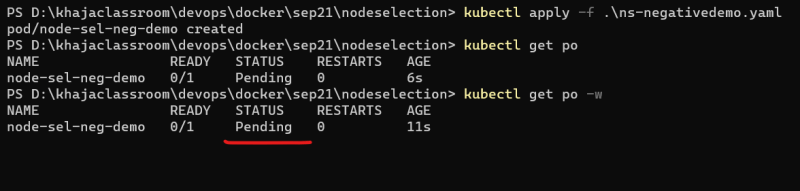
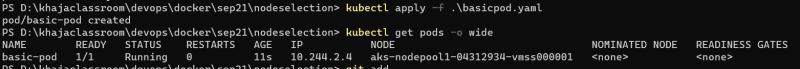
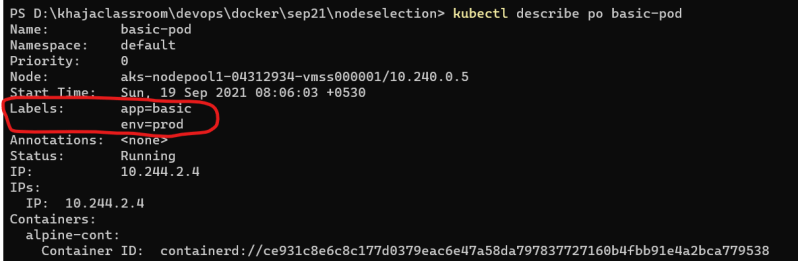
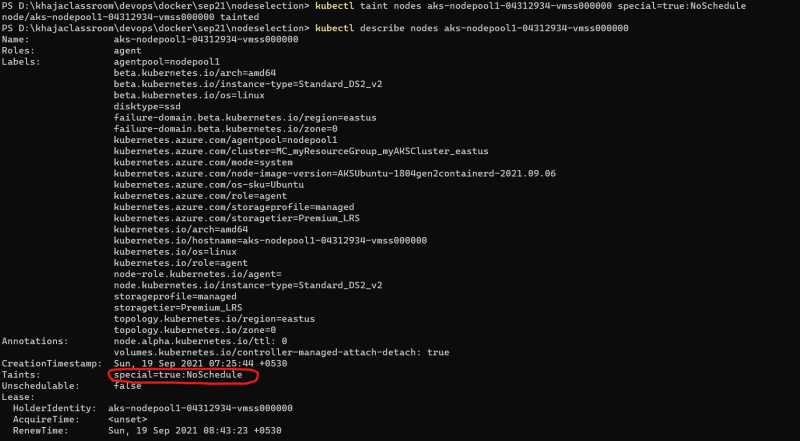
* [Refer Here](https://directdevops.blog/2019/11/02/deploying-the-docker-application-and-mysql-with-volume-support-into-kubernetes-from-code-to-docker-registries-like-acr-ecr-and-then-to-eks-aks/) for the article which is about the journey of application from containers to k8s

### Share this:

SEPTEMBER 19, 2021

# DevOps Classroom Series – 19/Set/2021

## Assigning Pods to nodes

* Generally scheduler will automatically do the reasonable plcement of your pods accross nodes, but there are some circumstances where you want to control which node the pod deploys
* Node Selector:
  + nodeSelctor is the simplest for node selection constraint.
  + This specifes a map of key value pairs.
  + For the pod to be elibigle to run on a node, node musht have indicated key value pairs as labels
  + Step 1: Attach label to nodes 
  + Step 2: Attach nodeSelector field to Pod Manifest [Refer Here](https://github.com/asquarezone/DockerZone/commit/6fa764f1282930c9754d2dc2c47f39b1e4fccad0)
  + Now apply the configuration and verify the pod schedule details 
  + Now lets try to do a negative scenario, lets try to use nodeSelector with labels that donot exist and see what happens
  + Pod spec [Refer Here](https://github.com/asquarezone/DockerZone/commit/971afc2a69fe06586b10af97e812b82a258642ad)
  + After applying we observe the pods will not be schedule by scheduler as no node has matching labels 
* Nodes in k8s come with pre-populate standard set of labels [Refer Here](https://kubernetes.io/docs/reference/labels-annotations-taints/)
* Affinity and anti-affinity:
  + The affinity/anti-affinity feature, greately expands types of constraints. The key enhancements from nodeSelector are
    - The affinity/anti-affinity language is more expressive, it offers more matching rules besides exact mathc created
    - You can indicate that a rule is soft or preference rather than hard requirement, so if the scheduler cannot statisfy, pods will still be scheduled
    - You can constraint aginst labels on other pods running on the node, rather than against labels of node itself, which allows rules about which pods can and cannot be located
  + Node affinity:
    - There are two types of node affinity
      * requiredDuringSchedulingIgnoredDuringExecution:
        + Hard requirement
      * prefferedDuringSchedulingIgnoredDuringExecution
        + Soft requirement
    - [Refer Here](https://github.com/asquarezone/DockerZone/commit/6ffe2d85cece08d16dc588a18b6d71d8073816f9) for the node affinity example
    - Created a basic pod with some labels [Refer Here](https://github.com/asquarezone/DockerZone/commit/385a64b41a8dc93489befd40ae1c04e312a7e47d)  
    - Write a Pod manifest
      * to schedule the new pod (pod-1) in the same node as basic-pod (podAffinity)
      * to schedule the new pod (pod-2) in the different node as basic-pod (podAntiAffinity)
* Taints and Tolerations:
  + Node affinity is property of Pods that attracts them to set of nodes.
  + Taints are opposite – they allow node to run a set of pods
  + Tolerations are applied to pods and allow the pods to schedule on the nodes with matching taints.
  + Taints and Tolerations work together to ensure pods are not scheduled into inappropriate nodes.
  + To apply a taint to the node kubectl taint nodes <node-name> key1=value1:NoSchedule
  + To remove the applied taint kubectl taint nodes <node-name> key1=value1:NoSchedule-
  + Lets apply a taint to any node 
  + Currently the following taints are supported
    - OutofDisk: node.kubernetes.io/out-of-disk
    - MemoryPressure: node.kubernetes.io/memory-pressure
    - DiskPressure: node.kubernetes.io/disk-pressure
    - PIDPressure: node.kubernetes.io/pid-pressure
  + Generally in some case it is advised to write tolerations based on node conditions
  + [Refer Here](https://github.com/asquarezone/DockerZone/commit/b064e8dad65ad5784f8970f2c8fd0ce941220fbe) for the yaml manifests

## Helm

* To install software’s in linux we use packaging tools such as apt, yum, in mac we have brew and in windows we have choco. These software are called as package managers
* Helm is an open source packaging tool for k8s to deploy and manage the lifecycle of your application
* Installing Helm [Refer Here](https://helm.sh/docs/intro/install/)
* In a very simple terms helm charts make the k8s manifest dynamic
* consider the following two deployments
* deployment 1

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: myapp-deploy

spec:

replicas: 2

selector:

matchLabels:

app: myapp

strategy:

type: RollingUpdate

rollingUpdate:

maxSurge: 50%

maxUnavailable: 50%

template:

metadata:

labels:

app: myapp

version: v3

spec:

containers:

- name: myapp-cont

image: shaikkhajaibrahim/myapp:3.0

ports:

- name: http

containerPort: 80

* the other deployment

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: otherapp-deploy

spec:

replicas: 3

selector:

matchLabels:

app: otherapp

strategy:

type: RollingUpdate

rollingUpdate:

maxSurge: 50%

maxUnavailable: 50%

template:

metadata:

labels:

app: otherapp

version: v1

spec:

containers:

- name: otherapp-cont

image: nginx

ports:

- name: http

containerPort: 80

protocol: TCP

* Helm use go templates to parametrize and value.yaml file which will have values defined
* Try installing any chart [Refer Here](https://bitnami.com/stacks/helm)

### Share this:

SEPTEMBER 19, 2021

# DevOps Classroom Series – 19/Sept/2021

## Docker compose

* Docker compose is a tool for defining and running multi container Docker applications
* Docker compose is written as a yaml file and [Refer Here](https://docs.docker.com/compose/compose-file/) for the yaml file reference
* create a yaml file as shown below

---

version: "3.8"

services:

web:

build: .

ports:

- 8080:8080

database:

image: mysql

environment:

MYSQL\_ROOT\_PASSWORD: directdevops

MYSQL\_DATABASE: test

MYSQL\_USER: directdevops

MYSQL\_PASSWORD: directdevops

volumes:

- type: volume

source: db\_data

target: /var/lib/mysql

volumes:

db\_data: {}

* Now docker-compose up -d will create images and start all the containers & docker-compose down will remove all the containers started.

## Docker on Windows

* From Windows 10 and Windows Server 2016 microsoft has added the support for running docker container natively on windows, prior to that on windows container were created using hypervisors (virtual box/hyperv)
* On Windows Systems, We can run docker containers with windows images as well as linux images,
* Windows containers cannot be run on the linux nodes
* Create a Windows Server and Enable the windows feature Containers
  + Launch Powershell as Admin and Execute
* Install-WindowsFeature -Name Containers
* UnInstall-WindowsFeature Windows-Defender
* Restart-Computer -Force
* Install Docker on Windows Server 2019 [Refer Here](https://docs.microsoft.com/en-us/virtualization/windowscontainers/quick-start/set-up-environment?tabs=Windows-Server#install-docker)
  + Launch Powershell as Admin and Execute
* Install-Module -Name DockerMsftProvider -Repository PSGallery -Force
* Install-Package -Name docker -ProviderName DockerMsftProvider
* Restart-Computer -Force
  + Login into the machine Launch Powershell as Admin again
* Start-Service Docker
  + pull some of the windows base images [Refer Here](https://hub.docker.com/_/microsoft-windows-base-os-images)
* docker pull mcr.microsoft.com/windows/servercore:ltsc2019
  + Lets quickly write one docker file to enable iis server with some html page using the above as base image

FROM mcr.microsoft.com/windows/servercore:ltsc2019

LABEL maintainer='khaja'

RUN dism.exe /online /enable-feature /all /featurename:iis-webserver /NoRestart

RUN echo "Hello World - Windows Container" > c:\inetpub\wwwroot\index.html

CMD ["cmd"]

* To deploy an iis application use <https://hub.docker.com/_/microsoft-windows-servercore-iis>
* [Refer Here](https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/adding-windows-nodes/) to add windows node to the k8s cluster
* [Refer Here](https://docs.microsoft.com/en-us/azure/aks/windows-container-cli) to add windows node to Azure AKS
* [Refer Here](https://docs.aws.amazon.com/eks/latest/userguide/windows-support.html) for window support on AWS EKS

### Share this:

### 