**Kubernetes**

kubectl run –replicas=1000 my-web-server

kubectl scale –replicas=2000 my-web-server

kubectl rolling-update my-web-server --image=web-server:2

kubectl rolling-update my-web-server –rollback

Kubernetes uses Docker host to host applications in the form of Docker containers (kubernetes supports rocket or crio)

Kubernetes cluster:

- Node it is a a machine phisycal or virtual on which the k8s software and set of tools are installed – worker machine where containers are launched

A cluster is a set of nodes grouped together

The master is a node with k8s control plane components installed and is responsible for the actual orchestration

Components:

- API Server – acts as the frontend for k8s – management devices, users, CLI – all talk to the API Server to interact with the cluster

- Etcd server – distributed reliable key value store used by k8s to store all data used to manage the – cluster – stores all the information on all the nodes in the cluster in a distributed manner – it is responsible for implementing logs within the cluster to ensure there are no conflicts between the masteers

- kubelet service – the agent that runs in each node in the cluster – responsible for making sure that the containers are running on the nodes as expected

- Container runtime (Docker) – is the underlying software that is used to run containers

- Controller – responsible for noticing and responding when nodes,containers or endpoints close down – makes decision to bring up new containers

- Scheduler – responsible for distributing work/containers across multiple nodes – it looks for newly created containers and assigns them to nodes

kubectl – command line utilty – kube control tool – it is the k8s CLI which is used to deploy and manage application on a k8s cluster, or to get cluster related information, to get the status of the nodes in the cluster etc

The kubectl run command is used to deploy an app on the cluster:

kubectl run <app>

The kubectl info viewer: kubectl cluster-info

Containers:

Docker

Rocket

Cri-O

Containerd

Container runtime interface (CRI) interface that allows any vendor to work as a container runtime for k8s as long as they adhered to the OCI standards

OCI = Open Container initiative and consists of an imagespec (specifications on how an image should be built) and a runtimespec (defined the standard on how any container runtime should be developed)

Docker does not meet the OCI standards, that in order for it to work dockershim was introduced.

Containerd was the container runtime daemon for docker, but now it s a container solution on it’s own (ctr is the command line for containerd, or nerdctl- better, even supports docker swarms, or crictl which is based on the CRI from k8s and is compatible with CRI container runtimes) – they are for debugging purposes, due to the fact that kubelet is not taking them into account and may delete the created containers with these commands

**Create an NGINX Pod**

kubectl run nginx --image=nginx

As of version 1.18, kubectl run (without any arguments such as --generator ) will create a pod instead of a deployment.

To create a deployment using imperative command, use **kubectl create:**

kubectl create deployment nginx --image=nginx

Kubernetes Concepts – <https://kubernetes.io/docs/concepts/>

Pod Overview- <https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/>

A k8s definition file always contains 4 top level fields (root level properties)

apiVersion: (the version of the k8s api we are using to create the object – it depends on what you would like to create)

- v1 for pods and services

- apps/v1 for ReplicaSet or Deployment

kind: (refers to the type of object we would like to create)

- Pod

- ReplicaSet

- Deployment

- Service

metadata: is data about the object, like name, labels etc

spec: is the specification area where we provide k8s with additional properties of that object (is different for different objects)

apiVersion: v1 – string

kind: Pod – string

metadata: - Dictionary

name: myapp-pod

labels:

app: myapp

type: front-end

tier: frontend

env: production

spec: -dictionary

containers: - List/Array (pods can have multiple containers)

- name: nginx-container – 1st item in the list – the item in the list is a dictionary

image: nginx

Create the pod: kubectl apply/create –f pod-definition.yaml

See the pod: kubectl get pods

Detailed info about the pod: kubectl describe pod <pod-name>

IDE which has free extensions for yaml validation for k8s is the Visual studio code ( YAML extension – made by RedHat)

You have to edit it first: enter the settings.json file and add:

{

”yaml.schemas”: {

”kubernetes”: ”\*.yaml”

}

}

ReplicaSet YAML file: - at the specification, as the replica set creates multiple instances of a pod, thus we have to create a template section under spec to be used by the replica set to create replicas

To populate the template, you must write a part of a pod manifest (metadata of the pod and the spec of that pod)

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: myapp-replicaset

labels:

app: myapp

type: front-end

**spec: (for the replicaset)**

**template:**

**metadata:**

**name: myapp-pod**

**labels:**

**app: myapp**

**type: front-end**

**tier: frontend**

**env: production**

**spec: (for the pod)**

**containers:**

**- name: nginx-container**

**image: nginx**

**replicas: 3**

**selector: (helps the replicaset what pods fall under it)**

**matchLabels:**

**type: front-end**

The replicaset is in fact a process which monitors the pods (based on LABELS) and when a pod fails it creates a new one

To fastly modify the number of replicas used (it DOES NOT modify the definition yaml file):

kubectl scale --replicas=n –f replicaset-definition.yml

or

kubectl scale --replicas=n replicaset(type) myapp-replicaset(name)

List replicasets: kubectl get replicaset

To replace or update the replicaset kubectl replace –f replicaset-definition.yaml

Run the command: You can check for apiVersion of replicaset by command kubectl api-resources | grep replicaset  
  
kubectl explain replicaset | grep VERSION and correct the apiVersion for ReplicaSet.

kubectl get all

To see revisions and rollouts of deploments:

kubectl rollout status deployment/<deployment\_name>

kubectl rollout histroy deployment/<dep\_name>

Rolling update is the default deploymet strategy in kuberentes

kuectl rollout undo deployment/<deployment\_name>

K8s when upgrading a deployment creates 2 replica sets and moves the pods to the new one to have no down time (it requires the killing of the old version pods). thus, the old replicaset will have 0 desired and 0 present pods and the new one will have 5 desired and 5 current.

To store informations about a rollout change:

kubectl edit deployment <name> --record