**Kubernetes for the Absolute Beginners – Hands-on**

The compatibility matrix issue is usually referred to as ‘The Matrix from Hell’.

Some of the different types of containers are LXC, LXD and LXCFS.

Docker utilizes LXC containers.

We won’t be able to run a **Windows** based container on a Docker host with Linux OS on it.

The main objective of Docker is to containerize applications and to ship them and run them.

Container runtime - Docker, Rocket, CRI-O

Container Orchestration Technologies :–

* Docker Swarm (from Docker)
* Mesos (Apache)
* K8S (from Google)

K8S - It is a container Orchestration technology used to orchestrate the deployment and management of containers in a clustered environment.

**K8S Architecture**

Nodes - A node is a physical or virtual machine on which K8S is installed. A node is a worker machine. It was also known as ‘minions’ in the past.

Cluster - It is a set of nodes grouped together.

Master - The Master is another node with K8S installed in it and is configured as a Master. The Master watches over the nodes in the cluster and is responsible for the actual orchestration of containers on the worker node.

Master actually manages the cluster, stores the information about members of the cluster, monitors the nodes, move the workload of failed node to another worker node.

**K8S Components**

API Server - acts as the front end for K8S. The users, management devices, CLI etc.. all talk to API Server.

etcd - It is a distributed reliable key value store used by K8S to store all data used to manage the cluster. Responsible for implementing locks within the cluster to ensure that there are no conflicts between the masters.

Scheduler - Responsible for distributing work or containers across multiple nodes.

Controller - Brain behind orchestration. Noticing and responding when nodes, containers or an end point goes down.

Container Runtime - Underlying software that is used to urn containers.

Kubelet - Agent that runs on each node in the cluster.

Containers are hosted in Nodes/minions and container runtime should be installed in Nodes.

Master server has ‘kube API Server’ which makes it a master. etcd, controller, and scheduler… installed on master

Nodes have ‘kubelet’ agent, responsible for interacting with a master. Also ‘container runtime’ is installed on Node.

[{Pic\_1 – Need to created and pasted here}]

‘kubectl’ command

kubectl run hello-minikube :- used to deploy an application on the cluster

kubectl cluster-info :- to view info about the cluster

kubectl get nodes :- to list all the nodes part of the cluster.

**Building K8S cluster**

Setup locally on laptop using :-

* Minikube (all in one server setup)
* MicroK8s
* kubeadm (multi-node configuration) – used to bootstrap and manage production grade K8S cluster.

Managed hosted Services :-

* GCP, AWS, Azure, IBM cloud, Digital Ocean, Oracle Cloud etc..

Kubectl get nodes –o wide (to get the host OS details)

Installing minikube in Oracle Virtual Box :-

* Follow the K8S official page to install minikube on Virtual Box
* Download ‘kubectl’ and ‘minikube’

**POD**

Containers are encapsulated into a K8S object knows as PODs. A POD is a single object of an application.

PODs have 1:1 relationship with the container running the application.

To scale up, we create new POD and to scale down we delete existing PODs.

We don’t add additional containers to existing POD to scale up the application.

A single POD can have multiple containers, but not same kind.

Multiple container inside the POD can communicate by referring local host since they share same network space and they easily share the same storage space as well.

Each PODs get its own IP address (internal IP) in K8S.

Deploy PODs

We can create PODs from CLI, using 2 ways :-

1. Create an NGINX POD (using --generator)

kubectl run --generator=run-pod/v1 nginx --image=nginx

1. Create an NGINX POD (using –restart=Never)

kubectl run nginx --image=nginx --restart=Never

If you run the **kubectl run**command without the ‘--restart=Never’ OR the ‘--generator=run-pod/v1’, the command will create a **deployment** instead.

To create a deployment

kubectl create deployment nginx --image=nginx

**YAML**

Data Structure Format - XML, JSON, YAML

The Equal number of spaces is very important in YAML to represent the data correctly.

* Key-value
* List

- Ordered collection

* Dictionary

- unordered collection

**POD with YAML**

Pod-definition.yml

K8S pod-definition file always contains 4 root level properties.

* **apiVersion:**

Version of K8S API using to create objects

|  |  |
| --- | --- |
| **Kind** | **Version** |
| POD | v1 |
| Service | v1 |
| ReplicaSet | apps/v1 |
| Deployment | apps/v1 |

* **kind:**

type of object we are trying to create

* **metadata:**

data about the object and it’s a dictionary.

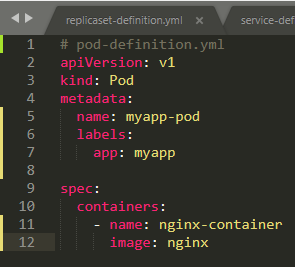
Labels in metadata will help to identify these objects at any point of time.

Under metadata, we can specify anything like name, labels etc.. that K8S expects. But under label ,we can have any key: value pair

* **spec:**

It is a dictionary of properties.

We define the containers to be included in the pod under ‘spec’



POD related Commands :-

kubectl get pods

kubectl get pods –o wide

kubectl describe pod <pod-name>

kubectl delete pod <pod-name>

{Create POD using --generator}

kubectl run --generator=run-pod/v1 nginx --image=nginx

{Create POD using --restart=Never}

kubectl run nginx --image=nginx --restart=Never

kubectl create -f <pod-definition file>

kubectl apply -f < pod-definition file >

**K8S Controllers**

Controllers are the brain behind the K8S

They are the processes that monitor K8S objects and respond accordingly

Replication Controllers:

Helps to run multiple instances of a single POD in K8S cluster, to provide HA

It always ensure specified number of containers are running at all times.

It helps to share the load across multiple PODs

Replication controller spans across multiple nodes in the cluster

Replica Set:

Replication controller is the old technology and its being replaced by ‘ReplicaSet’

Replication Controller definition file

It also has the same 4 root level properties {apiVersion, kind, metadata, spec}

‘kind’ would be ReplicationController OR ReplicaSet

The ‘template’ section under ‘spec’, we define the POD definition.

We define the number of replicas required in ‘replicas’ property

If we are defining ReplicaSet, the apiVersion could be ‘apps/v1’.

ReplicaSet requires ‘selector’ definitions under ‘spec’. It helps the ReplicaSet to identify what PODs are falling under it.

‘selector’ is also applicable for ReplicationController. If we don’t define ‘selector’ in the definition file, it assumes it to be the same as the labels provided in the POD definition.

In case of ReplicaSet, a user input is required for this property.

ReplicaSet and RplicationController will always monitor the PODs with the same label, and if any fails deploy the new one to meet the required number of replicas defined in the definition file.

Replica related Commands :-

kubectl create -f replicaset-definition.yml

kubectl get replicaset

kubectl delete replicaset <replicaset-name>

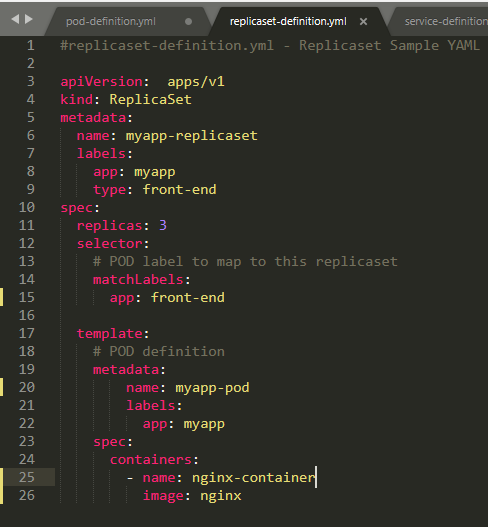
kubectl describe replicaset <replicaset-name>

kubectl edit replicaset <replicaset-name>

kubectl replace -f replicaset-definition.yml

kubectl scale --replicas=3 -f replicaset-definition.yml

kubectl get all {to get all object}



**Deployments**

Deploying application in production environment

We have to Deploy, Upgrade the new version, perform rolling upgrades, Roll back/undo the recent changes, pause and resume to apply the changes.

Deployments will make all these happen.

Deployment is K8S object comes higher in hierarchy.

Deployment-definition file

It is exactly similar to the replica set definition file, except for the kind, which is ‘Deployment’.

Deployment Commands

kubectl create -f deployment-definition.yml

kubectl create -f deployment-definition.yml --record

kubectl describe deployment

kubectl get deployments

kubectl delete deployment

kubectl describe deployment

kubectl apply -f deployment-definition.yml

kubectl run nginx --image=nginx {not using definition file}

Rolling Upgrades : updating the newer version one after another, without impacting the user.

**Deployments – Update & Rollback**

If we first create a deployment, it triggers a ‘Rollout’ and new ‘Rollout’ creates a new deployment Revision.

When the application is upgraded, a new rollout is triggered and a new deployment Revision is created.

Deployment Strategy

There are 2 types of deployment strategies:

Recreate strategy - The process of destroying the older version all together and redeploy the newer version together. This cause an service interruption in between.

Rolling Updates - In this strategy, it will destroy the older version and deploy the newer version one by one. Make sure that there is no service interruption.

Rolling Updates is the default deployment strategy in K8S

Rollout Commands

kubectl rollout status <deployment-name>

kubectl rollout history <deployment-name>

kubectl rollout undo <deployment-name>

kubectl apply –f <deployment-definition file>

kubectl set image deployment/myapp-deployment nginx=nginx:1.9.1

**Networking in K8S**

In K8S IP address is assigned to POD and the internal IP range is 10.244.0.0

When K8S is initially configured it creates an internal private network with the address 10.2440.0 and all the PODs are attached to it.

These IPs are dynamic, it changes when PODs are recreated.

K8S does not automatically setup any kind of networking to handle the IP conflict issue when multiple nodes are part of the cluster.

All containers/PODs must be able to communicate to one another without having to configure NAT.

All nodes must be able to communicate with all containers and vice-versa without NAT.

There are multiple pre-build solutions available for the criteria.

Like CISCO SCI network, cilium, Big Cloud Fabric, flannel, vmwareNSX, calico ..

**Services**

K8S services enable communication between various components within and outside of the application.

It enables coupling between micro-services in our application.

Service Types

NodePort Service - is a K8S object and its use case is to listen to a port on the node and forward request on that port on to the POD.

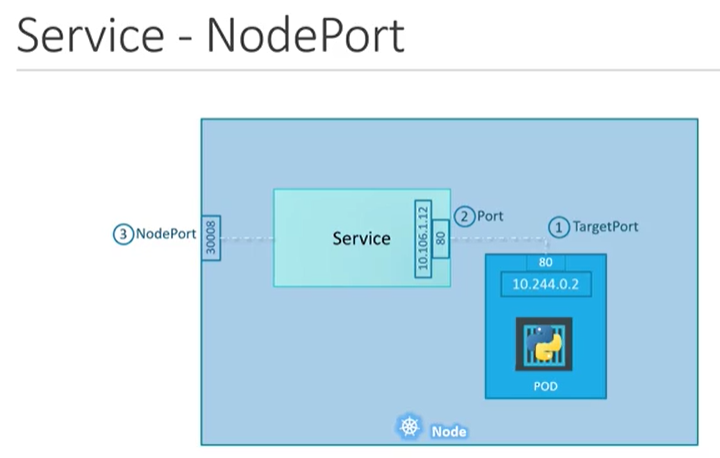
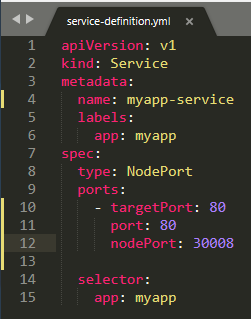
ClusterIP - It creates a virtual IP inside the cluster to enable communication between different services.

LoadBalancer -

NodePort

* TargetPort
* Port
* NodePort

The valid range of NodePort is from 30000 to 32767

service-definition file

‘port’ in the ‘ports’ property is mandate one

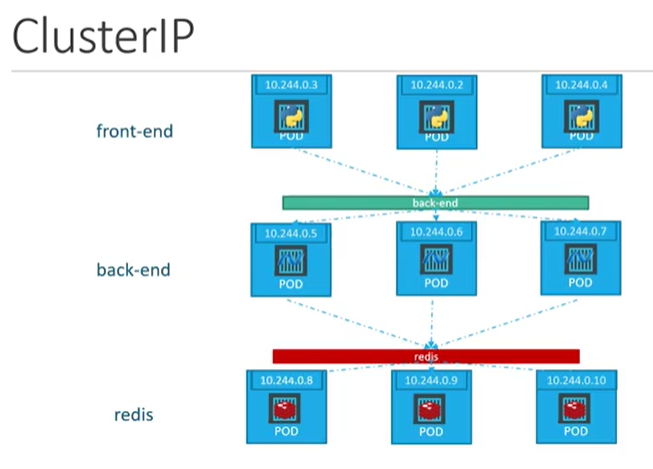
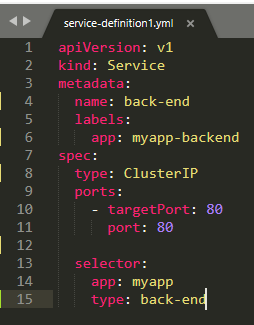
If don’t provide targetPort it is assumed to be the same as port.

If don’t provide nodePort, a Freeport in the valid range is automatically allocate

clusterIP

We cannot rely on the dynamic IP address of the POD for internal communication.

A K8S service can help to group the PODs together and provide a single interface to access the PODs in a group.

Service Commands

kubectl create -f service-definition.yml

kubectl get services

**References**

K8S website - <https://kubernetes.io/>

For Lab Setup :-

Oracle Virtual Box – <https://www.virtualbox.org/>

For OS – <https://www.osboxes.org/>

Minikube :-

Install MiniKube - <https://kubernetes.io/docs/tasks/tools/install-minikube/>

MiniKube Download Windows - <https://github.com/kubernetes/minikube/releases>

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

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

Answer to all exercises - <https://github.com/mmumshad/kubernetes-training-answers>