**Day 20**

**=============================================================================**

**Kubernetes**

**===================**

**Menions: This is an individual node used in kubernetes**

**Combination of these minions is called as Kubernetes cluster**

**Master is the main machine which triggers the container orchestraion**

**It distributes the work load to the Slaves**

**Slaves are the nodes that accept the work load from the master**

**and handle activites load balancing,autoscalling,high availability etc**

**==============================================**

**Kubernetes unmanaged setup on Centos**

**=========================================**

**Install, start and enable docker service**

**yum install -y -q yum-utils device-mapper-persistent-data lvm2 > /dev/null 2>&1**

**yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo > /dev/null 2>&1**

**yum install -y -q docker-ce >/dev/null 2>&1**

**systemctl start docker**

**systemctl enable docker**

**=====================================================================================**

**Disable SELINUX**

**setenforce 0**

**sed -i --follow-symlinks 's/^SELINUX=enforcing/SELINUX=disabled/' /etc/sysconfig/selinux**

**============================================================================================**

**Disable SWAP**

**sed -i '/swap/d' /etc/fstab**

**swapoff -a**

**===========================================================================================**

**Update sysctl settings for Kubernetes networking**

**cat >>/etc/sysctl.d/kubernetes.conf<<EOF**

**net.bridge.bridge-nf-call-ip6tables = 1**

**net.bridge.bridge-nf-call-iptables = 1**

**EOF**

**sysctl --system**

**============================================================================================**

**Add Kubernetes to yum repository**

**cat >>/etc/yum.repos.d/kubernetes.repo<<EOF**

**[kubernetes]**

**name=Kubernetes**

**baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86\_64**

**enabled=1**

**gpgcheck=1**

**repo\_gpgcheck=1**

**gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg**

**https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg**

**EOF**

**======================================================================================**

**Install Kubernetes**

**yum install -y kubeadm kubelet kubectl**

**==================================================================================**

**Enable and start Kubernetes service**

**systemctl start kubelet**

**systemctl enable kubelet**

**=====================================================================================**

**Repeat the above steps on Master and slaves**

**=======================================================================================**

**On Master=============**

**===========**

**Initilise the Kubernetes cluster**

**-----------------------------------------**

**kubeadm init --apiserver-advertise-address=ip\_of\_master --pod-network-cidr=192.168.0.0/16**

**=========================================================================================**

**To be able to use kubectl command to connect and interact with the cluster,**

**the user needs kube config file.**

**mkdir /home/centos/.kube**

**cp /etc/kubernetes/admin.conf /home/centos/.kube/config**

**chown -R centos:centos /home/centos/.kube**

**========================================================================================**

**Deploy calico network**

**kubectl create -f https://docs.projectcalico.org/v3.9/manifests/calico.yaml**

**========================================================================================**

**For slaves to join the cluster**

**kubeadm token create --print-join-command**

**======================================================================================**

**Check the pods of kube-system are running**

**kubectl get pods -n kube-system**

**============================================================================**

**Day 21**

**=========================================================================**

**Kubernetes**

**======================**

**Menions: This is an individual node used in kubernetes**

**Combination of these minions is called as Kubernetes cluster**

**Master is the main machine which triggers the container orchestraion**

**It distributes the work load to the Slaves**

**Slaves are the nodes that accept the work load from the master**

**and handle activites load balancing,autoscalling,high availability etc**

**Kubernetes uses various of types of Object**

**1 Pod: This is a layer of abstraction on top of a container.This is the samallest**

**object that kubernetes can work on.In the Pod we have a container.**

**The advantage of using a Pod is that kubectl commands will work on the Pod and the**

**Pod communicates these instructions to the container.In this way we can use the**

**same kubectl irresepective of which technology containers are in the Pod.**

**2 Service: This is used for port mapping and network load balancing**

**3 NameSpace: This is used for creating partitions in the cluster.Pods running**

**in a namespace cannot communicate with other pods running in other namespace**

**4 Secrets: This is used for passing encrypted data to the Pods**

**5 ReplicationController: This is used for managing multiple replicas of PODs**

**and also perfroming saclling**

**6 ReplicaSet: This is similar to replicationcontroller but it is more advanced**

**where features like selector can be implemented**

**7 Deployment: This used for perfroming all activites that a Replicaset can do**

**it can also handle rolling update**

**8 Volume: Used to preserve the data even when the pods are deleted**

**9 Statefulsets: These are used to handle stateful application like data bases**

**where consistency in read write operations has to be maintained.**

**10 Ingress: This object is used for mapping ip with domain name**

**Kubernetes Architecture**

**=============================**

**Master Componentes**

**=======================**

**Container runtime: This can be docker or anyother container technology**

**apiServer: Users interact with the apiServer using some clinet like ui,command line tool like kubelet.It is the apiServer which is the gateway to the cluster**

**It works as a gatekeeper for authentication and it validates if a specific**

**user is having permissions to execute a specific command.Example if we want to**

**deploy a pod or a deployment first apiServers validates if the user is authorised to perform that action and if so it passes to the next process**

**ie the "Scheduler"**

**Scheduler: This process accepts the instructions from apiServer after validation**

**and starts an application on a sepcific node or set of nodes.It estimates**

**how much amount of h/w is required for an application and then checks which**

**slave have the necessary h/w resources and instructs the kubelet to deploy**

**the application**

**kubelet: This is the actual process that takes the orders from scheduler and**

**deploy an application on a slave.This kubelet is present on both master and slave**

**controller manager: This check if the desired state of the cluster is always**

**maintained.If a pod dies it recreates that pod to maintain the desired state**

**etcd: Here the cluster state is maintained in key value pairs.**

**It maintains info about the slaves and the h/w resources available on**

**the slaves and also the pods running on the slaves**

**The scheduler and the control manager read the info from this etcd**

**and schedule the pods and maintain the desired state**

**===========================================================================**

**Worker components**

**=======================**

**containerrun time: Docker or some other container technology**

**kubelet: This process interacts with container run time and the node**

**and it start a pod with a container in it**

**kubeproxy: This will take the request from services to pod**

**It has the intellegence to forward a request to**

**a near by pod.Eg If an application pod wants to communicate with a db pod**

**then kubeproxy will take that request to the nearby pod**

**=================================================================================**

**=============================================================================**

**Setup of ManagedKubernetes**

**===============================**

**Free**

**===========**

**1 http://katakoda.com**

**(or)**

**2 http://playwithk8s.com**

**Paid**

**==============**

**1 Signup for a Google cloud account**

**2 Click on Menu icon on top right corner--->Click on Kubernetes Engine-->Clusters**

**3 Click on Create cluster--->Click on Create**

**================================================================================**

**Day 22**

**================================================================================**

**=========================================================================**

**UseCase**

**===========**

**Create nginx as a pod and name it webserver**

**kubectl run --image nginx webserver**

**To see the list of pods running**

**kubectl get pods**

**To see more info about the pods like their ip and slave where they are running**

**kubectl get pods -o wide**

**To delete the pod**

**kubectl delete pods webserver**

**============================================================================**

**UseCase**

**=========**

**Create mysql pod and name it mydb and go into its interactive terminal and create few tables**

**kubectl run --image mysql:5 mydb --env MYSQL\_ROOT\_PASSWORD=intelliqit**

**To check the pods**

**kubectl get pods**

**To go into the interactive terminal**

**kubectl exec -it mydb -- bash**

**To login into the db**

**mysql -u root -p**

**Password: intellqiit**

**Create tables here**

**=========================================================================**

**Kuberentes Defintion files**

**==============================**

**Objects in Kubernetes cluster are deployed using these**

**defintion files**

**They are created using yml and they generally these 4 top level**

**fields.**

**apiVersion:**

**kind:**

**metadata:**

**spec:**

**apiVersion : This specifies the code library that has to be imported**

**to create a particualr kind of Kubernetes object**

**kind: Here we specify the type kubernetes object that we want to**

**create(Pod,ReplicaSet,Deployment,Service etc)**

**metadata: Here we can give additional info about the Pod like**

**the name of the Pod,some labels etc**

**spec: This is where exact info about the object that is created is**

**specified like containers info port mapping,no of replicas etc**

**================================================================**

**kind apiVersions**

**===================================================**

**Pod v1**

**Service v1**

**Secret v1**

**Namespace v1**

**ReplicationController v1**

**Volume v1**

**ReplicaSet apps/v1**

**Deployment apps/v1**

**StatefuleSet apps/v1**

**==================================================================**

**Create a pod defintion file to start nginx pod with a name webserver**

**1 vim pod-defintion1.yml**

**---**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: nginx-pod**

**labels:**

**type: proxy**

**author: intelliqit**

**spec:**

**containers:**

**- name: webserver**

**image: nginx**

**...**

**2 Create pod from the above file**

**kubectl apply -f pod-defintion1.yml**

**3 To check the list of pods**

**kubectl get pods**

**4 To delete the pods**

**kubectl delete -f pod-defintion1.yml**

**========================================================================**

**UseCase**

**================**

**Create a postgres-pod and give the labels as author=intelliqit**

**and type=db,also pass the necessay environment variables**

**1 vim pod-definition2.yml**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: postgres-pod**

**labels:**

**author: intelliqit**

**type: db**

**spec:**

**containers:**

**- name: mydb**

**image: postgres**

**env:**

**- name: POSTGRES\_PASSWORD**

**value: intelliqit**

**- name: POSTGRES\_USER**

**value: myuser**

**- name: POSTGRES\_DB**

**value: mydb**

**...**

**To create pods from the above file**

**kubectl apply -f pod-defintion2.yml**

**====================================================================**

**UseCase**

**============**

**Create a jenkins-pod and also perfrom necessary port mapping**

**vim pod-definition2.yml**

**---**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: jenkins-pod**

**labels:**

**type: ci-cd**

**author: intelliqit**

**spec:**

**containers:**

**- name: jenkins**

**image: jenkins/jenkins**

**ports:**

**- containerPort: 8080**

**hostPort: 8080**

**...**

**To create the pods from the above file**

**kubectl apply -f pod-defintion3.yml**

**To check if the jnekins pod is running**

**kubectl get pods -o wide**

**To accesss jenkins from browser**

**kubectl get nodes -o wide**

**Capture the external ip of the node where jenkins pod is running**

**in browser**

**externalip:8080**

**============================================================================**

**Day 23**

**=============================================================================**

**========================================================================**

**ReplicationController**

**=======================**

**This is a high level Kubernets object that can be used for handling**

**multiple replicas of a Pod.Here we can perfrom Load Balancing**

**and Scalling**

**ReplicationController uses keys like "replicas,template" etc in the "spec" section**

**In the template section we can give metadata related to the pod and also use**

**another spec section where we can give containers information**

**Create a replication controller for creating 3 replicas of httpd**

**vim repilication-controller.yml**

**---**

**apiVersion: v1**

**kind: ReplicationController**

**metadata:**

**name: httpd-rc**

**labels:**

**author: intelliqit**

**spec:**

**replicas: 3**

**template:**

**metadata:**

**name: httpd-pod**

**labels:**

**author: intelliqit**

**spec:**

**containers:**

**- name: myhttpd**

**image: httpd**

**ports:**

**- containerPort: 80**

**hostPort: 8080**

**...**

**To create the httpd replicas from the above file**

**kubectl create -f replication-controller.yml**

**To check if 3 pods are running an on whcih slaves they are running**

**kubectl get pods -o wide**

**To delete the replicas**

**kubectl delete -f replication-controller.yml**

**ReplicaSet**

**===================**

**This is also similar to ReplicationController but it is more**

**advanced and it can also handle load balancing and scalling**

**It has an additional field in spec section called as "selector"**

**This selector uses a child element "matchLabels" where the**

**it will search for Pod based on a specific label name and try to add**

**them to the cluster**

**Create a replicaset file to start 4 tomcat replicas and then perform scalling**

**vim replica-set.yml**

**---**

**apiVersion: apps/v1**

**kind: ReplicaSet**

**metadata:**

**name: tomcat-rs**

**labels:**

**type: webserver**

**author: intelliqit**

**spec:**

**replicas: 4**

**selector:**

**matchLabels:**

**type: webserver**

**template:**

**metadata:**

**name: tomcat-pod**

**labels:**

**type: webserver**

**spec:**

**containers:**

**- name: mywebserver**

**image: tomcat**

**ports:**

**- containerPort: 8080**

**hostPort: 9090**

**To create the pods from the above file**

**kubectl create -f replica-set.yml**

**Scalling can be done in 2 ways**

**a) Update the file and later scale it**

**b) Scale from the coomand prompt withbout updating the defintion file**

**a) Update the file and later scale it**

**Open the replicas-set.yml file and increase the replicas count from 4 to 6**

**kubectl replace -f replicas-set.yml**

**Check if 6 pods of tomcat are running**

**kubectl get pods**

**b) Scale from the coomand prompt withbout updating the defintion file**

**kubectl scale --replicas=2 -f replica-set.yml**

**================================================================**

**Deployment**

**================**

**This is also a high level Kubernetes object which can be used for**

**scalling and load balancing and it can also perfrom rolling update**

**Create a deployment file to run nginx:1.7.9 with 3 replicas**

**vim deployment1.yml**

**---**

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: nginx-deployment**

**labels:**

**author: intelliqit**

**type: proxyserver**

**spec:**

**replicas: 3**

**selector:**

**matchLabels:**

**type: proxyserver**

**template:**

**metadata:**

**name: nginx-pod**

**labels:**

**type: proxyserver**

**spec:**

**containers:**

**- name: nginx**

**image: nginx:1.7.9**

**ports:**

**- containerPort: 80**

**hostPort: 8888**

**To create the deployment from the above file**

**kubectl create -f deployment.yml**

**To check if the deployment is running**

**kubectl get deployment**

**To see if all 3 pod of nginx are running**

**kubectl get pod**

**Check the version of nginx**

**kubectl describe pods nginx-deployment | less**

**==========================================================================**

**Namespace in kubernetes**

**==========================**

**Namespaces are used to create partitions in the Kubernetes cluster**

**Pods runnign in different namespaces cannot communicate with**

**each other**

**To create Namespaces**

**===========**

**vim namespace.yml**

**---**

**apiVersion: v1**

**kind: Namespace**

**metadata:**

**name: test-ns**

**...**

**kubectl apply -f namespace.yaml**

**To see the list of namespace**

**================================**

**kubectl get namespace**

**Create a pod on that namespace**

**===================================**

**vim pod-definition4.yml**

**---**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: jdk-pod**

**namespace: test-ns**

**labels:**

**author: intelliqit**

**spec:**

**containers:**

**- name: java**

**image: openjdk:12**

**...**

**To see list of pods in a namespace**

**======================================**

**kubectl get pods -n test-ns**

**To delete a namespace**

**===========================**

**kubectl delete namespace test-ns**

**==========================================================================**

**Day 24**

**==========================================================================**

**Kompose**

**================**

**This is used to implement docker compose to create a multi**

**container architecture in Kubernetes**

**Implementing docker compose can be done using Kompose**

**docker compose + docker swarm = docker stack**

**docker compose + Kubernetes = Kompose**

**Setup**

**===========**

**1 Download Kompose**

**curl -L https://github.com/kubernetes/kompose/releases/download/v1.18.0/kompose-linux-amd64 -o kompose**

**2 Give execute permissions**

**chmod +x kompose**

**3 Move it to PATH**

**sudo mv ./kompose /usr/local/bin/kompose**

**4 To check if the installion is successfull**

**kompose version**

**Digital Ocean URL**

**========================**

**https://www.digitalocean.com/community/tutorials/how-to-migrate-a-docker-compose-workflow-to-kubernetes**

**Create a docker compose file**

**vim docker-compose.yml**

**---**

**version: '3'**

**services:**

**mydb:**

**image: mysql:5**

**environment:**

**MYSQL\_ROOT\_PASSWORD: intelliq**

**wordpress:**

**image: wordpress**

**ports:**

**- 8080:80**

**deploy:**

**replicas: 3**

**...**

**To setup the above architecture in Kubernetes**

**kompose up**

**To create kubernetes definition files**

**kompose convert**

**To delete the above create architecture**

**kompose down**

**Note: Practice Kompsoe on katokoda.com**

**======================================================================**

**---**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: redis-pod**

**labels:**

**author: intelliqit**

**spec:**

**containers:**

**- name: redis**

**image: redis**

**volumeMounts:**

**- name: redis-volume**

**mountPath: /data/redis**

**volumes:**

**- name: redis-volume**

**emptyDir: {}**

**Create a pod from the above file**

**kubectl create -f volumes.yml**

**To check if the volume is mounted**

**kubectl exec -it redis-pod -- bash**

**Go to the redis folder and create some files**

**cd redis**

**cat > file**

**Store some data in this file**

**To kill the redis pod install procps**

**apt-get update**

**apt-get install -y procps**

**Identify the process id of redis**

**ps aux**

**kill 1**

**Check if the redis-pod is recreated**

**kubectl get pods**

**We will see the restart count changes for this pod**

**If we go into this pods interactive terminal**

**kubectl exec -it redis-pod -- bash**

**We will see the data but not the s/w's (procps) we installed**

**cd redis**

**ls**

**ps This will not work**

**============================================================================**

**Day 25**

**============================================================================**

**==============================================================**

**Service Object**

**=====================**

**This is used for network load balancing and port mapping**

**It uses 3 ports**

**1 target port: Pod or container port**

**2 port: Service port**

**3 hostPort: Host machines port to make it accessable from external network**

**Service objects are classified into 3 types**

**1 clusterIP: This is the default type of service object used in**

**Kubernetes and it is used when we want the Pods in the cluster to**

**communicate with each other and not with extrnal networks**

**2 nodePort: This is used if we want to access the pods from an extrnal**

**network and it also performs network load balancing ie even if a pod**

**is running on a specific salve we can access it from other slave in**

**the cluster**

**3 LoadBalancer: This is similar to Nodeport and it is used for external**

**connectivity of a Pod and also network load balancing and it also assigns**

**a public ip for all the slave combined together**

**=============================================================================**

**Use Case**

**=================**

**Create a service defintion file for port mapping an nginx pod**

**vim pod-defintion1.yml**

**---**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: nginx-pod**

**labels:**

**author: intellqit**

**type: proxy**

**spec:**

**containers:**

**- name: appserver**

**image: nginx**

**=========================================================**

**vim service1.yml**

**---**

**apiVersion: v1**

**kind: Service**

**metadata:**

**name: nginx-service**

**spec:**

**type: NodePort**

**ports:**

**- targetPort: 80**

**port: 80**

**nodePort: 30008**

**selector:**

**author: intellqit**

**type: proxy**

**Create pods from the above pod definition file**

**kubectl create -f pod-definition1.yml**

**Create the service from the above service definition file**

**kubectl create -f service.yml**

**Now nginx can be accesed from any of the slave**

**kubectl get nodes -o wide**

**Take the external ip of any of the nodes:30008**

**UseCase**

**==================**

**Create a pod defintion file to start a ghost pod and also create a**

**service object of the type LoadBalancer**

**vim pod-defintion7.yml**

**---**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: ghost-pod**

**labels:**

**author: intelliqit**

**type: CMS**

**spec:**

**containers:**

**- name: ghost**

**image: ghost**

**...**

**vim service2.yml**

**---**

**apiVersion: v1**

**kind: Service**

**metadata:**

**name: ghsot-service**

**labels:**

**author: intelliqit**

**spec:**

**type: LoadBalancer**

**ports:**

**- targetPort: 2368**

**port: 2368**

**selector:**

**type: CMS**

**author: intelliqit**

**=====================================================================**

**UseCase**

**================**

**Create a pod-definiton file for httpd pod and create a service object**

**of type cluster ip for it**

**vim pod-definition8.yml**

**---**

**apiVersion: v1**

**kind: Service**

**metadata:**

**name: ghsot-service**

**labels:**

**author: intelliqit**

**spec:**

**type: LoadBalancer**

**ports:**

**- targetPort: 2368**

**port: 2368**

**selector:**

**type: CMS**

**author: intelliqit**

**...**

**vim service3.yml**

**---**

**apiVersion: v1**

**kind: Service**

**metadata:**

**name: httod-service**

**labels:**

**author: intelliqit**

**spec:**

**ports:**

**- targetPort: 80**

**port: 80**

**selector:**

**author: intelliqit**

**type: webserver**

**...**

**=========================================================================**

**UseCase**

**==============**

**Create a deployment file of for tomcat and also create a servcie file**

**of type node port**

**vim deployment3.yml**

**---**

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: tomcat-deployment**

**labels:**

**type: appserver**

**spec:**

**replicas: 2**

**selector:**

**matchLabels:**

**type: appserver**

**template:**

**metadata:**

**name: tomcat-pod**

**labels:**

**type: appserver**

**spec:**

**containers:**

**- name: tomcat**

**image: tomee**

**...**

**vim service4.yml**

**---**

**apiVersion: v1**

**kind: Service**

**metadata:**

**name: tomcat-service**

**labels:**

**author: intelliqit**

**spec:**

**type: NodePort**

**ports:**

**- targetPort: 8080**

**port: 8080**

**selector:**

**type: appserver**

**...**

**==========================================================================**

**===========================================================================**

**Day 26**

**===========================================================================**