**Kubernetes regular commands**

When you install Kubernetes on a System, you are actually installing the component like

**API Server.**

The API server acts as the front-end for kubernetes. The users, management devices, Command line interfaces all talk to the API server to interact with the kubernetes cluster.

**ETCD service.**

Next is the ETCD key store. ETCD is a distributed reliable key-value store used by kubernetes to store all data used to manage the cluster. Think of it this way, when you have multiple nodes and multiple masters in your cluster, etcdstores all that information on all the nodes in the cluster in a distributed manner. ETCD is responsible for implementing locks within the cluster to ensure there are no conflicts between the Masters.

**kubeletservice.**

Kubelet is the agent that runs on each node in the cluster. The agent is responsible for making sure that the containers are running on the nodes as expected.

**Container Runtime,**

The container runtime is the underlying software that is used to run containers. In our case it happens to be Docker.

**Controllers,**

The controllers are the brain behind orchestration. They are responsible for noticing and responding when nodes, containers or endpoints goes down. The controllers makes decisions to bring up new containers in such cases.

**Schedulers.**

The scheduler is responsible for distributing work or containers across multiple nodes. It looks for newly created containers and assigns them to Nodes.

**Master and Worker** and a set of components that make up Kubernetes. But how are these components distributed across different

The **worker node** (or minion) as it is also known, is were the containers are hosted. For example Docker containers, and to run docker containers on a system, we need a container runtime installed. And that’s were the container runtime falls. In this case it happens to be Docker. This doesn’t HAVE to be docker, there are other container runtime alternatives available such as Rocket or CRIO.

The **master server** has the kube-apiserver and that is what makes it a master.

Similarly, the worker nodes have the kubeletagent that is responsible for interacting with the master to provide health information of the worker node and carry out actions requested by the master on the worker nodes.

All the information gathered are stored in a key-value store on the Master. The key value store is based on the popular etcdframework as we just discussed.

The master also has the controller manager and the scheduler.

Kubectl run nginx —image=nginx

Kubectl get pods

Kubectl describe nginx

Kubectl create -f deployment.ymal —record

Kubectl get deployments

Kubectl apply -f deployment —record

Kubectl set image deployment container=nginix:1.18

Kubectl rollout status deployment

Kubectl rollout history deployment

Kubectl rollout undo deployment —record

A cluster is a set of resources, worker nodes, networks, and storage devices that keep apps highly available. After you have your cluster, then you can deploy your apps in containers.

* PID: process IDs
* USER: user and group IDs
* UTS: host name and domain name
* NS: mount points
* NET: network devices, stacks, and ports
* CGROUPS: control limits and monitoring of resources

Kubernetes Resources

* **Config maps**: holds configuration data for pods to consume
* **Daemon sets**: ensures that each node in the cluster runs this pod
* **Deployments**: defines a desired state of a deployment object
* **Events**: provides life cycle events on pods and other deployment objects
* **Endpoints:** allows an inbound connections to reach the cluster services
* **Ingress**: a collection of rules that allows inbound connections to reach the cluster services
* **Jobs**: creates one or more pods and when they complete successfully, the job is marked as completed
* **Node:** a worker machine in Kubernetes
* **Namespaces**: multiple virtual clusters backed by the same physical cluster
* **Pods**: the smallest deployable units of computing that can be created and managed in Kubernetes
* **Persistent volumes**: provides an API for users and administrators to abstract details about how storage is provided from how it is consumed
* **Replica sets**: ensures that a specified number of pod replicas are running at any given time
* **Secrets**: holds sensitive information, such as passwords, OAuth tokens, and SSH keys
* **Service accounts**: provides an identity for processes that run in a pod
* **Services**: an abstraction that defines a logical set of pods and a policy by which to access them, sometimes called a microservice
* **Stateful sets**: the workload API object that manages stateful applications

Linux namespaces

PID

MNT

NET

IPC

User

UTC

**Nodes (or) minions**

A Cluster is a set of nodes (Mater & Worker Nodes)

Components—> API, etc, kubelet(agent), Container runtime, Scheduler, Controller.

Command line utility kubectl, kube command line, or tube control.

Kubectl run command is used to deploy an application on the cluster .

The “kubectl cluster-info” command is used to view information about the cluster.

**Minikube**——> VirtualBox(hypervisor)

Is a iso image

kubectl get nodes

kubectl get nodes -o wide

Kubectl run nginx —image=nginx

Kubectl get pods

Kubectl describe nginx

Kubectl apply -f pod.yaml

Kubectl describe pod nginx

Kubectl edit pod nginx

Pod definition file

***Replication Controller and ReplicaSets:***

- Replication Controller is older technology that is replaced by replicates, replica set is new recommendation to set replication

Kubectl create -f rc-definition.yaml

Kubectl get replication controller

Kubectl get pods

***Replica Set:***

Kubectl create -f replicates-definition.yaml

Kubectl get replicates

Kubectl get pods

***Labels & Selectors:***

***Scale:***

Kubectl replace -f replicaset-definition.yaml (or)

Kubectl scale —replica=6 -f replicates-definition.yaml (or)(this will not change in definition file)

Kubectl scale —replica=6 -f replicates myapp-replicaset

***Deployments:***

Kubectl create -f deployment.yaml

Kubetctl get deployments

Kubectl get all

Kubectl describe deployment my app-deployment

***Rollout and Versioning***

Rollout Command

Kubectl rollout status deployment/myapp-deployment

Kubectl rollout history deployment/myapp-deployment

Rollback

Kubectl rollout undo deployment/myapp-deployment

***Deployment Strategy***

*Recreate startegy, Rolling Update(default)*

**Kubectl apply**

Kubectl apply -f deployment-definition.yml (or)

Kubectl set image deployment/myapp-deployment \ nginx=nginx:1.9.1

**Record change-cause**

kubectl create -f deployment.yaml --record

kubectl rollout history deployment/myapp-deployment

***Networking in Kubernetes***

IP address is assigned to a POD (Ex: 10.244.0.2)

Internal private network address is created when kuberenets is initially configured on each node.

Cluster Networking

Routing techniques(NSX,vmware,cilium,flannel & cisco)

***Services -***

Kubernetes services helps us connect applications together with other applications or users

***NodePort, clusterIP, LoadBalancer***

Nodeport- 30000 to 32767 (port on a node to pod)

clusterIP- Services creates virtual IP inside cluster to enable communication between services such as set of front end servers to backend servers

Loadbalancer- Where it provisions a load balancer for our application in supported cloud providers

TargetPort(pod port)—>port(Service port)——>Nodeport(port on node)

List services

Kubectl get services

kubectl describe service kubernetes

Default Service is ClusterIP

Loadbalcers like Jproxy or nginx

**IBM kubernetes**

All Redis Pods running or not

date; hostname -f;kubectl get pods --all-namespaces=true | grep -i redis

Synthetic failure for Orient\_me

[kubestarterpub.g3.conncloudk8s.com](http://kubestarterpub.g3.conncloudk8s.com)

kubectl get namespaces

kubectl get pods --all-namespaces=true | grep -i -v running

List of pods

kubectl -n connections get pods -o wide

Kubectl —n name get pods -0 wide

To restart pods first tree commands

kubectl delete pods -n=connections orient-web-client-b4d5744fb-srrtb

To get logs from pods

kubectl logs -n=connections orient-web-client-b4d5744fb-bff9p

Get logs of a pod in a log file

kubectl logs -n=connections orient-web-client-b4d5744fb-bff9p > a.log

Get last 100 lines of log

tail -100 a.log

scp soumith@kubestarterpub.a3.conncloudk8s.com:/home/SSO/soumith/a.log ./

Check the status of pods

kubectl get pods -n connections | grep redis

Which Elastic pods is not running

kubectl get po -n connections | grep -i elastic

kubectl -n connections delete pod mw-elasticsearch-master-0

Deleting which pod is not running

kubectl get po -n connections | grep -i elasticsearch-master

Namespace

kubectl get ns

Connections from ladp server

Netstat -n | grep ldap

Get BSS rest pods

kubectl get pods -n bss | grep -i rest

Get Memory usage of each pod

kubectl top pod -n bss (rb-db-rest-pod id)

Delete pods one at a time

kubectl -n bss delete pod (rb-db-rest-pod id)

All pods

kubectl get pods —all-namespaces

Kubernetes starts with three initial namespaces:

* **default** The default namespace for objects with no other namespace
* **kube-system** The namespace for objects created by the Kubernetes system
* **kube-public** This namespace is created automatically and is readable by all users (including those not authenticated). This namespace is mostly reserved for cluster usage, in case that some resources should be visible and readable publicly throughout the whole cluster. The public aspect of this namespace is only a convention, not a requirement.

**IBM cloud**

**1)Ibm Cloud login from CLI using federated access**

Ibmcloud login —sso

2)to create a free cluster in ibmcloud

ibmcloud ks cluster create classic --name my\_cluster

ibmcloud ks clusters is to make sure that your cluster is in Normal state

ibmcloud ks workers <yourclustername> and make sure that all workers are in Normal state with a Ready status.

3)Create a namespace in IBM Cloud Container Registry where you can store your images:

ibmcloud cr namespace-add soumithregistry

ibmcloud cr namespace-list

4)Build the example Docker image:

docker build --tag us.icr.io/soumithregistry/hello-world .

docker image ls

5)Push the image to IBM Cloud Container Registry:

docker push [us.icr.io/soumithregistry/hello-world](http://us.icr.io/soumithregistry/hello-world)

6)To set the environment variable and download the Kubernetes configuration files

ibmcloud ks cluster-config <cluster\_name\_or\_ID>

7)To set the path to the local Kubernetes configuration file as an environment variable

**export KUBECONFIG=/Users/Soumith.Kumar.Nimmagadda@ibm.com/.bluemix/plugins/container-service/clusters/soumithcluster/kube-config-hou02-soumithcluster.yml**

8)Run image as a deployment

kubectl run hello-world --image=us.icr.io/<namespace>/hello-world

To check the status of your deployment

kubectl get pods

9)When the status reads Running, expose that deployment as a service, which is accessed through the IP of the worker nodes. The example for this lab listens on port 8080. Run this command:

kubectl expose deployment/hello-world --type="NodePort" --port=8080

10)Find the port that is used on that worker node and examine your new service:

kubectl describe service <name-of-deployment>

11)Run the following command and note the public IP as <public-IP>.

ibmcloud cs workers <name-of-cluster>

You can now access your container/service by using curl <public-IP>:<nodeport> (or a web browser). You're done if you see this:

“Hello world! Your app is up and running in a cluster!”

**Scale apps with replicas**

12)Update the replica set:

kubectl edit deployment/<name-of-deployment>

13)Edit the YAML file in the code editor of your choice.

14)Change the replicas number from 1 to 10 so that the configuration now reads:

15)Watch your changes being rolled out:

kubectl rollout status deployment/<name-of-deployment>

16)After the rollout is finished, ensure your pods are running:

kubectl get pods

Update and roll back apps

17)Before you begin, ensure that you have the image tagged with 1 and pushed:

docker build --tag us.icr.io/<namespace>/hello-world:1 .

docker push us.icr.io/<namespace>/hello-world:1

To update and roll back:

* Make a change to your code and build a new docker image with a new tag: docker build --tag us.icr.io/<namespace>/hello-world:2 .
* Push the image to the IBM Cloud Container Registry: docker push us.icr.io/<namespace>/hello-world:2

18)Using kubectl, update your deployment to use the latest image. You can do this in two ways:

1)Edit the YAML file again by using kubectl edit deployment/<name-of-deployment>

2)Specify a new image by using a single command.

kubectl set image deployment/hello-world hello-world=us.icr.io/<namespace>/hello-world:2

19)Check he status of the rollout by running one of these commands:

* 1. kubectl rollout status deployment/<name-of-deployment>
  2. kubectl get replicasets

20)Confirm that your new code is active: curl <public-IP>:<nodeport> **Optional**: Undo your latest rollout:kubectl rollout undo deployment/<name-of-deployment>

Check the health of app