**Kubernetes**

Kubernetes in an open source container management tool hosted by Cloud Native Computing Foundation (CNCF). This is also known as the enhanced version of Borg which was developed at Google to manage both long running processes and batch jobs, which was earlier handled by separate systems.

Kubernetes comes with a capability of automating deployment, scaling of application, and operations of application containers across clusters. It is capable of creating container centric infrastructure.

**Features of Kubernetes**

Following are some of the important features of Kubernetes.

* Continues development, integration and deployment
* Containerized infrastructure
* Application-centric management
* Auto-scalable infrastructure
* Environment consistency across development testing and production
* Loosely coupled infrastructure, where each component can act as a separate unit
* Higher density of resource utilization
* Predictable infrastructure which is going to be created

One of the key components of Kubernetes is, it can run application on clusters of physical and virtual machine infrastructure. It also has the capability to run applications on cloud. It helps in moving from host-centric infrastructure to container-centric infrastructure.

**Kubernetes - Images**

Kubernetes (Docker) images are the key building blocks of Containerized Infrastructure. As of now, we are only supporting Kubernetes to support Docker images. Each container in a pod has its Docker image running inside it.

When we are configuring a pod, the image property in the configuration file has the same syntax as the Docker command does. The configuration file has a field to define the image name, which we are planning to pull from the registry.

Following is the common configuration structure which will pull image from Docker registry and deploy in to Kubernetes container.

apiVersion: v1

kind: pod

metadata:

name: Tesing\_for\_Image\_pull -----------> 1

spec:

containers:

- name: neo4j-server ------------------------> 2

image: <Name of the Docker image>----------> 3

imagePullPolicy: Always ------------->4

command: ["echo", "SUCCESS"] ------------------->

In the above code, we have defined −

* **name: Tesing\_for\_Image\_pull** − This name is given to identify and check what is the name of the container that would get created after pulling the images from Docker registry.
* **name: neo4j-server** − This is the name given to the container that we are trying to create. Like we have given neo4j-server.
* **image: <Name of the Docker image>** − This is the name of the image which we are trying to pull from the Docker or internal registry of images. We need to define a complete registry path along with the image name that we are trying to pull.
* **Image Pull Policy** − Always - This image pull policy defines that whenever we run this file to create the container, it will pull the same name again.
* **command: [“echo”, “SUCCESS”]** − With this, when we create the container and if everything goes fine, it will display a message when we will access the container.

In order to pull the image and create a container, we will run the following command.

$ kubectl create –f Tesing\_for\_Image\_pull

Once we fetch the log, we will get the output as successful.

$ kubectl log Tesing\_for\_Image\_pull

**Kubernetes - Namespace**

Namespace provides an additional qualification to a resource name. This is helpful when multiple teams are using the same cluster and there is a potential of name collision. It can be as a virtual wall between multiple clusters.

Functionality of Namespace

Following are some of the important functionalities of a Namespace in Kubernetes −

* Namespaces help pod-to-pod communication using the same namespace.
* Namespaces are virtual clusters that can sit on top of the same physical cluster.
* They provide logical separation between the teams and their environments.

**Create a Namespace**

The following command is used to create a namespace.

apiVersion: v1

kind: Namespce

metadata

name: elk

Control the Namespace

The following command is used to control the namespace.

$ kubectl create –f namespace.yml ---------> 1

$ kubectl get namespace -----------------> 2

$ kubectl get namespace <Namespace name> ------->3

$ kubectl describe namespace <Namespace name> ---->4

$ kubectl delete namespace <Namespace name>

# Kubernetes - Service

A service can be defined as a logical set of pods. It can be defined as an abstraction on the top of the pod which provides a single IP address and DNS name by which pods can be accessed. With Service, it is very easy to manage load balancing configuration. It helps pods to scale very easily.

A service is a REST object in Kubernetes whose definition can be posted to Kubernetes apiServer on the Kubernetes master to create a new instance.

Service without Selector

apiVersion: v1

kind: Service

metadata:

name: Tutorial\_point\_service

spec:

ports:

- port: 8080

targetPort: 31999

The above configuration will create a service with the name Tutorial\_point\_service.

Service Config File with Selector

apiVersion: v1

kind: Service

metadata:

name: Tutorial\_point\_service

spec:

selector:

application: "My Application" -------------------> (Selector)

ports:

- port: 8080

targetPort: 31999

# Kubernetes - Labels & Selectors

**Labels**

Labels are key-value pairs which are attached to pods, replication controller and services. They are used as identifying attributes for objects such as pods and replication controller. They can be added to an object at creation time and can be added or modified at the run time.

**Selectors**

Labels do not provide uniqueness. In general, we can say many objects can carry the same labels. Labels selector are core grouping primitive in Kubernetes. They are used by the users to select a set of objects.

Kubernetes API currently supports two type of selectors −

* Equality-based selectors
* Set-based selectors

**Equality-based Selectors**

They allow filtering by key and value. Matching objects should satisfy all the specified labels.

Set-based Selectors

Set-based selectors allow filtering of keys according to a set of values.

apiVersion: v1

kind: Service

metadata:

name: sp-neo4j-standalone

spec:

ports:

- port: 7474

name: neo4j

type: NodePort

selector:

app: salesplatform ---------> 1

component: neo4j -----------> 2