**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.

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.
* **imagePullPolicy** − 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.

**Kubernetes 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.

**Kubernetes 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.

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>

In the above code,

* We are using the command to create a namespace.
* This will list all the available namespace.
* This will get a particular namespace whose name is specified in the command.
* This will describe the complete details about the service.
* This will delete a particular namespace present in the cluster.

**Kubernetes Node:**

A node is a working machine in Kubernetes cluster which is also known as a minion. They are working units which can be physical, VM, or a cloud instance.

Each node has all the required configuration required to run a pod on it such as the proxy service and kubelet service along with the Docker, which is used to run the Docker containers on the pod created on the node.

They are not created by Kubernetes but they are created externally either by the cloud service provider or the Kubernetes cluster manager on physical or VM machines.

**Service with Selector**

apiVersion: v1

kind: node

metadata:

name: < ip address of the node>

labels:

name: <lable name>

In JSON format the actual object is created which looks as follows −

{

Kind: node

apiVersion: v1

"metadata":

{

"name": "10.01.1.10",

"labels"

{

"name": "cluster 1 node"

}

}

}

**Node Controller**

They are the collection of services which run in the Kubernetes master and continuously monitor the node in the cluster on the basis of metadata.name. If all the required services are running, then the node is validated and a newly created pod will be assigned to that node by the controller. If it is not valid, then the master will not assign any pod to it and will wait until it becomes valid.

Kubernetes master registers the node automatically, if **–register-node** flag is true.

–register-node = true

However, if the cluster administrator wants to manage it manually then it could be done by turning the flat of −

–register-node = false