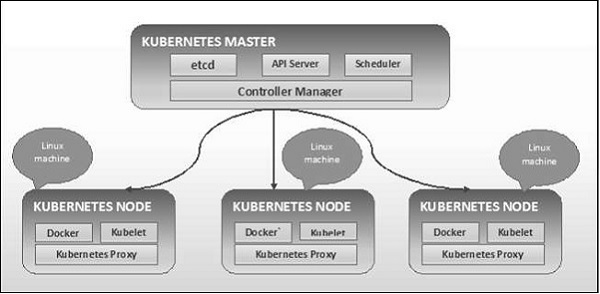
**Kubernetes:**

**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.

## Kubernetes - Cluster Architecture

As seen in the following diagram, Kubernetes follows client-server architecture. Wherein, we have master installed on one machine and the node on separate Linux machines.



## Kubernetes - Master Machine Components

### **etcd**

It stores the configuration information which can be used by each of the nodes in the cluster.

### **API Server**

Kubernetes is an API server which provides all the operation on cluster using the API. API server implements an interface, which means different tools and libraries can readily communicate with it.

### **Controller Manager**

This component is responsible for most of the collectors that regulates the state of cluster and performs a task. In general, it can be considered as a daemon which runs in nonterminating loop and is responsible for collecting and sending information to API server.

### **Scheduler**

This is one of the key components of Kubernetes master. It is a service in master responsible for distributing the workload.

## Kubernetes - Node Components

### **Docker**

The first requirement of each node is Docker which helps in running the encapsulated application containers in a relatively isolated but lightweight operating environment.

### **Kubelet Service**

This is a small service in each node responsible for relaying information to and from control plane service. It interacts with **etcd** store to read configuration details and wright values.

### **Kubernetes Proxy Service**

This is a proxy service which runs on each node and helps in making services available to the external host.

**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.

**Services:** 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.

**Pod:** A pod is a collection of containers and its storage inside a node of a Kubernetes cluster. It is possible to create a pod with multiple containers inside it

**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.

**Containerization:** Kubernetes containerization is the utilization of the Kubernetes open source tool to automate the deployment, scaling, and management of containers without launching virtual machines for any applications.

**Kubernetes Image:**

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.

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 Jobs:** The main function of a job is to create one or more pod and tracks about the success of pods. They ensure that the specified number of pods are completed successfully. When a specified number of successful run of pods is completed, then the job is considered complete.

We will create the job using the following command with yaml which is saved with the name **py.yaml**.

$ kubectl create –f py.yaml

The above command will create a job. If you want to check the status of a job, use the following command.

$ kubectl describe jobs/py

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

**Replication Controller:**

Replication Controller is one of the key features of Kubernetes, which is responsible for managing the pod lifecycle. It is responsible for making sure that the specified number of pod replicas are running at any point of time.

**Replica Set:**

Replica Set ensures how many replica of pod should be running. It can be considered as a replacement of replication controller. The key difference between the replica set and the replication controller is, the replication controller only supports equality-based selector whereas the replica set supports set-based selector.

**Deployments:**

Deployments are upgraded and higher version of replication controller. They manage the deployment of replica sets which is also an upgraded version of the replication controller. They have the capability to update the replica set and are also capable of rolling back to the previous version.

**Volumes:**

In Kubernetes, a volume can be thought of as a directory which is accessible to the containers in a pod. We have different types of volumes in Kubernetes and the type defines how the volume is created and its content.

The concept of volume was present with the Docker, however the only issue was that the volume was very much limited to a particular pod. As soon as the life of a pod ended, the volume was also lost

**Secrets:**

Secrets can be defined as Kubernetes objects used to store sensitive data such as user name and passwords with encryption.

There are multiple ways of creating secrets in Kubernetes.

* Creating from txt files.
* Creating from yaml file.

**Diff b/w Docker and K8S:**

|  |  |
| --- | --- |
| **Docker** | **Kubernetes** |
| 1. Docker has the ability to reduce the size of development by providing a smaller footprint of the operating system via containers. | * Continues development, integration and deployment * Containerized infrastructure |
| 2.With containers, it becomes easier for teams across different units, such as development, QA and Operations to work seamlessly across applications. | * Application-centric management * Auto-scalable infrastructure |
| 3.You can deploy Docker containers anywhere, on any physical and virtual machines and even on the cloud. | * Environment consistency across development testing and production * Loosely coupled infrastructure, where each component can act as a separate unit |
| 4.Since Docker containers are pretty lightweight, they are very easily scalable. | * Higher density of resource utilization * Predictable infrastructure which is going to be created |