Kubernetes is the go-to platform for hosting production grade application,

Kubernetes or K8s, was originally designed by Google in the year of 2014, now it is taken care by CNCF (Cloud Native Computing Foundation) based on their experience running containers in Production environment, it is now an open-source project, it is arguably one of the best and most popular container orchestration technology. This tool written in Golang (Google Language).

Before understanding Kubernetes, we need to understand what is Container + Orchestration,

Let’s understand 1st what is container by using the most popular tool “Docker”

[ex: Create an end-to-end stack including various different technologies, like a “webserver using NodeJS”, “Database MongoDB”, “Messaging system like Redis”, “Orchestration tool like Ansible”, when developing Application using these tools we may come across with a lot issues, (like, Compatibility with OS version, because a lot of times versions of the tools and OS are not going to be compatible – at this time we may have to change the OS that supports for all these tools compatibility; 2nd we need to check the compatibility between the services with libraries & dependencies on the OS, at this times also we will face a lot of issues where one service may require a specific dependent library and another may require a different one, not only this even the application architecture are going to change overtime after a lot these changes; The bigger issue is after changing anything we need to go through the entire process of checking the compatibility between the various components and underlying infrastructure.

**Why do we need container?**

To avoid Compatibility/Dependency

To avoid Long setup time

Different Dev/Test/Prod environments

Containers can help us with the compatibility issue, it will allow to modify to change the components without affecting the other components, and even modify the underlying operating system as required, and by using Docker we can run each component in a separate container with its own libraries and dependencies, to make this possible we need to configure Docker once. Irrespective of the underlying OS they run just by installing Docker on top this, we can bring our Containers.

**What are Containers?**

In simple, Containers are isolated environments as in they can have their own processes, services, networks just like Virtual machines except they all share the same OS Kernel.

Operating System:

Let’s try to understand about OS 1st, every OS got 2 components like Kernel and Software; where Kernel responsible for interacting with the Hardware and Software consists of different user interface, drivers, compilers, file managers, developer tools etc., while the OS kernel the same (Linux) but the software on top of it makes the OS different, in this scenario we have the common Linux kernel shared across all OS and some custom software that differentiate OS from each other.

Sharing the Kernel:

Let’s see with an example: we have a system with OS (Ubuntu) on it, where Docker can run any flavor of OS on top of it as long as they all based on the same kernel (in this case “Linux”), now each container is going to have only the additional software that makes this OS different and Docker utilizes the underlying Kernel of Docker Host which works with the all the OS above. If you are trying to use Windows here that is not going to work, because this is not a Hypervisor where we can virtualize run different OS & kernels on the same Hardware. The main purpose of Docker is it containerize application, and to ship & run them.

**Difference between Virtual Machines & Containers:**

In Docker, we have the Hardware infrastructure, then the OS, and Docker installed on the OS, after this setup Docker is going to manage Containers that run with libraries & dependencies alone. Docker are light-weighted one where only MB’s are used, this leads to boot up the Docker faster within few secs.

On the other side, Virtual machine we have the Hardware Infra, then OS and we Hypervisor installed where we need to have different Virtual machines installed (Guest OS) to manage the setup and then we have the dependencies and the Applications, this causes higher utilization of underlying resources as there are multiple OS and Kernels running. VM’s are usually heavy weight as GB’s, this leads to few mints to boot up each VM’s.

**How Docker works?**

As we can see the most common images are readily available within Docker Hub/Registry by pulling that from there by simple command **(docker pull nginx)** to our host then proceed with the Container setup **(docker run nginx).**

An Image is a package or template just like a VM template, it is used to create 1 or more Docker Containers, whereas, containers are running instances of Images that are isolated and have their own environments and processes. As we can see most of the images are already available but if we don’t have one, then we can create our own Image by writing a Dockerfile and pushing it to the Docker Registry and making it available for the public.

**Container Orchestration:**

Now consider we already have our application into our Docker Container, but what’s next? How to deploy this into Production? What if our Application relies on other container? Like Databases, Messaging services, or other backend services. What if the number of users increase and if we need to scale our application? How to scale down whenever the user decreases? To enable these functionalities we need to have an underlying platform with a set of resources. This platform needs to orchestrate the connectivity between the containers and takes care of scaling based on the load, these process of automatically deploying and managing containers are known as Container Orchestration.

To do this Orchestration we do have multiple tools in the market like Kubernetes (Google), Docker Swarm (Docker), MesOS (Apache), etc.,

When comes to Docker Swarm it’s easy to setup and get started it lacks some of the advanced features required for complex applications.

MesOS on the other hard difficult to setup and get started but supports many advanced features.

Whereas Kubernetes are the most popular of all these all, is a bit difficult to setup and started but provides a lot of options to customize deployments and supports deployments of complex architectures. Kubernetes is supported to all public cloud service providers (like AWS, Azure, GCP etc.,)

**Advantages of Container Orchestration:**

Consider our application is highly available and hardware failure should not bring our application down, as multiple instances of our applications running on different nodes, the user traffic balanced across various containers when demand increases deploy more instances of the application within a matter of seconds, and we have the ability to do the same on service level when we run out of hardware resources scale the number of nodes accordingly without application downtime, all this are easily done by setting up an declarative object configuration files, and this is Kubernetes; it is a container orchestration technology where used to deploy and manage 100s and 1000s of containers in a clustered environment.

**Kubernetes Architecture:**

1. ***Nodes*** == A node is a machine (physical or virtual) on which kubernetes is installed, a node is a worker machine that is where containers are launched. It was also known as (Minions) on the past. But what if the node where our application runs fails? Obviously our application goes down? That is why we are maintaining multiple nodes for application management.
2. ***Cluster*** == A cluster is a set of nodes group together, this way even one node fails we have other nodes to manage our application. Moreover, having multiple nodes helps in sharing load as well.
3. ***Master*** == now we have a cluster, but who is responsible for managing the cluster? Where is the information about the members in the cluster stored? How are the nodes monitored? When a node fails how do you move the workload of the worker node? That’s where the master comes in, Master is another node with kubernetes installed in it and configured as a master. The master watches over the nodes in the cluster and it’s responsible for the actual orchestration of containers on the worker nodes.

**Components:**

When we install kubernetes on a system, we are actually installing the following components (API server, etcd, Kubelet, Container Runtime, Controller, and Scheduler)

***API server*** == acts as the frontend for kubernetes (users, management devices, CLI) all talk to the API server to interact with the kubernetes cluster

***Etcd*** == is a distributed reliable key-value store, to store all data in order to manage the cluster. (ex: when we have multiple nodes and masters in a clusters, etcd will store all those data’s in a distributed manner; this responsible for implementing logs within the cluster to ensure that there are no conflicts between the masters)

***Scheduler*** == is responsible for distributing works for containers across multiple nodes, this looks for the newly created containers and assigns them to nodes.

***Controller*** == are the brain behind orchestration, they responsible for noticing and responding when nodes, containers and endpoints goes down, the controllers makes decision to bring new containers in such cases.

***Container Runtime*** == software used to run containers (in our case it’s docker) but there are other options as well (like rkt (rocket), CRI-O)

***Kubelet*** == agent runs on each nodes in the cluster, this agent responsible for making sure that the containers are running on the nodes as expected.

***Kubectl:***

Kube command line tool (or) kubectl (or) kube control, used to deploy and manage applications on a kubernetes cluster using commands. This tool used to communicate with the clusters.

Kubectl command format,

Kubectl <operation\_command> <type\_command> <name\_command>

Operations: get, create, delete, describe etc.,

Type: pod, deployment, jobs, namespace etc.,

Name: search-pod, cart-deployment, ui-service, mail-jobs etc.,

Eg: kubectl <get> <pod> <test-pod>

**Types of Kubernetes installation:**

Kubernetes being a complex container orchestration platform, offers various installation methods to cater to different use cases, environments, and preferences.

1. ***Kubernetes distribution-specific installation tools:***

* Kubeadm == this tool for setting up a cluster easily on machines running system. It’s part of the kubernetes project and is widely used for setting up clusters.
* Minikube == useful for single-node cluster, mostly used for testing phase
* Kops (kubernetes operations) == This tool helps in setting up production-grade clusters on AWS
* K3s == A lightweight kubernetes distribution suitable for IoT, edge and development environments.

1. ***Package Managers:***

* Helm == used to install kubernetes applications and manage their dependencies. Helm charts provide a way to define, install, and upgrade even complex kubernetes applications.

1. ***Manual Installation:***

* By downloading Kubernetes binaries, configuring them manually, and setting up each component.

1. ***Cloud Providers:***

* EKS (Elastic Kubernetes Services), AKS (Azure Kubernetes Services) GKE (Google Kubernetes Engine) etc., abstract away the complexities of cluster management, making it easy to deploy and manage clusters in the cloud.

1. ***Containerized Installation:***

* Kubespray (or) KubeADM-Ansible == allows you to deploy kubernetes clusters using containers, making it easier to manage dependencies and ensuring consistent deployments across different environments.

1. ***Turnkey Solutions:***

* Rancher (or) Openshift == provides kubernetes along with additional features and tools for easier management and deployment of containerized applications.

1. ***Community Projects:***

* K8s-Saltstack, K8s-Deployer, Kargo == Various community-driven projects and scripts exist that help in deploying kubernetes clusters

Each of these methods has its own advantages and suitability depending on factors like environment (local, cloud, on-prem), scale, level of customization required, and familiarity with the tools. It’s essential to choose the right method based on your specific requirements and constraints.

Reference Link: <https://kubernetes.io/docs/concepts/>