**What is Kubernetes?**

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services—with a framework to run distributed systems resiliently. It takes care of your scaling requirements, failover, deployment patterns, scaling, load balancing, logging, and monitoring, much like PaaS offerings. However, it operates at the container level rather than at the hardware level.

It was initially built upon a decade and a half of the Google experience running production workloads. Open-sourced in 2014, Kubernetes is now a growing ecosystem that combines best practices for application deployment to run some of the largest software services by scale.

The name Kubernetes is derived from a Greek term meaning ‘helmsman’ or ‘pilot.’ True to this word, Kubernetes provides the guiding force for developer platforms to transition from virtual machines (VMs) to containers and the statically scheduled to the dynamically scheduled. This means no more manual integration and configuration when you move from a testing environment to an actual production environment or from on-premise to the cloud! The Kubernetes logical compute environment offers common services to all the applications in the cluster as part of the ecosystem for the software to run consistently.

**What can you do with Kubernetes?**

Developers need to write applications that can run across multiple operating environments. Traditionally, the performance of these applications is dependent on a particular infrastructure, such as adherence to cloud provider-specific constructs and back-end storage systems. Kubernetes eliminates the need for one infrastructure lock-in by allowing you to deploy cloud-native applications on containers without any restrictions and manage them exactly as you like anywhere you want.

**How Does Kubernetes Work?**

Before we dive into how Kubernetes works, you need to first understand what containers are and why people use them. A container is a small, lightweight virtual machine (VM) that does not have device drivers and shares its operating system among the applications. It is a good way to bundle and run applications in a production environment. However, you need to manage these containers in a proper way so that there is no downtime. This is where Kubernetes comes to the rescue.



Fig: Traditional vs Virtualized vs Container Development

Kubernetes works as a “container orchestration system” that manages the lifecycle of containerized applications and automates the deployment of several containers. Containers running the same applications are usually grouped together into Pods. There is a dedicated container orchestrator which supervises these groups and ensures that they operate correctly.

**Why do You Need Kubernetes?**

Containers decompose applications into smaller parts and enable faster development by assigning smaller, more focused teams responsible for specific containers. However, it requires a proper system for integrating and orchestrating each of these smaller decomposed parts. Kubernetes makes this possible by introducing Pods, or a collection of containers.

Fig: Kubernetes architecture

There can be one or multiple containers in a single Pod and each of them shares the same IP address and resources such as memory and storage. By grouping the containers in this manner, Kubernetes eliminates the need to cram multiple functionalities in one single container.

**Why is Kubernetes so Popular?**

Kubernetes’ rise in popularity is due to its successful use cases in mission-critical sectors such as banking, edtech, and IT. With more and more enterprises adopting hybrid cloud strategies, there has been a number of challenges in the field of application development. Kubernetes is the perfect choice for companies that use hybrid clouds because of its consistency in both public and on-premise clouds.

**Features of Kubernetes**

Kubernetes offers the widest range of features required to deploy containerized applications.

**1. Portable and Open-Sourced**

As an open-source platform, Kubernetes can run containers on any number of public clouds, virtual machines, or infrastructures. Its compatibility with most platforms makes it highly flexible and usable.

**2. Programming Language and Framework Support**

Kubernetes supports most programming languages and frameworks.

**3. Automatic Resource Bin Packing**

The application is packaged, and the containers scheduled based on available resources, allowing optimal utilization of unused resources. As Kubernetes enables you to specify the CPU and RAM needs of each container, the containers can be slotted to increase compute efficiency and ultimately lowers costs.

**4. Container Deployment Control**

Kubernetes allows complete control over the number of containers you want with deployment and keeps those containers ready with a rollout. Thus, you can automate Kubernetes to create new containers, remove existing containers, or adopt all of their resources to a new container.

**5. Automated Rollouts and Rollbacks**

Versions and updates are automated and running, so you don’t waste time or resources on downtime. Also, the health of the application is screened during rollout to automatically rollback in the case of any glitch or failure.

**6. Health Checks and Self-healing**

It checks the health of nodes and containers to ensure than an application doesn’t fail. In case of a pod crash or an error, Kubernetes automatically restarts containers that fail, replaces or kills containers that don’t match user-defined health checks, and doesn’t make them available to clients until they are client-ready.

**7. Secure Configuration Management**

You can store and manage user information such as passwords and SSH keys, deploy secrets and application configuration without rebuilding your container images, and do all of this without exposing secrets in your stack configuration.

**8. Service Discovery and Load Balancing**

Kubernetes can expose a container using the DNS or IP address. For high traffic to a container, it can automatically balance the loads into the pods and distribute the network traffic for the stable deployment of software.

This supports the distribution of load and auto-balancing of resources instantly during incidental traffic or batch processing.

**9. Storage Orchestration**

You can automatically mount a storage system or orchestrate containers on multiple hosts.

**10. Auto-Scaling of Resources and Applications in Real-Time**

Kubernetes offers several features for auto-scaling. You can deploy and control the number of containers based on computing resources, workload balance, and scale-out your software or create applications on more containers by grouping containers in pods. Horizontal autoscaling is another feature whereby Kubernetes auto-scalers automatically size a deployment’s number of pods based on the usage of specified resources and at the individual server level.

New servers can be added or removed easily. Kubernetes can thus automatically expose your containers to the internet or other containers in the cluster to automatically load balance traffic across matching containers.

**11. Heterogeneous Clusters**

Kubernetes allows you to build your cluster with a mix of virtual machines on the cloud, on-premise, or in your data center, to suit your requirements.

**12. Persistent Storage Support**

Kubernetes workflow includes support for Amazon Web Services EBS, Google Cloud Platform persistent disks, and other storage.

**13. Workload Support**

Kubernetes supports a variety of workloads: stateless, stateful, data-processing.

**14. Application Type Support**

Kubernetes offers complete support for the application types, application frameworks, and language without differentiating between apps and services.

**Benefits of Using Kubernetes**

Although we have several tools in DevOps that are equally popular like the Docker, Kubernetes wins the votes. This is because of the many benefits that far outweigh other tools.

Among its many attributes, Kubernetes:

Lays the foundations for developing and building cloud-native applications that can run anywhere, independent of cloud requirements

Speeds up the process of building, testing, and releasing software

Has the ability to handle scaling-up of both applications and infrastructure in real-time

Tackles workload scalability on the fly

Controls resource consumption and hardware use

Balances application load across the host infrastructure

Moves an application to another host in the event of resource shortage

Facilitates easy rollbacks

Tests and auto-corrects applications

Delivers the software quickly with better compliance

Increases transparency and collaboration within the teams and pipelines

Effectively minimizes security risk while controlling cost

Increases the efficiency of server usage

Renders health-check of your apps and self-heals with auto-placement, auto-restart, auto-replication, and auto-scaling

Can be combined with other open-source projects to orchestrate all parts of your container infrastructure

Supports better IT security

Helps manage your containerized applications more easily and quickly

Increases developer productivity

Automates patches and updates

Allows visibility for in-process and failed deployments with status query support

Saves time when a deployment is paused at any time, to be quickly resumed later

Allows version control with newer versions of application images or a rollback when the current version is not stable

Supports container balancing as it automatically places containers by computing the best location

Manages your batch and compute-intensive (CI) workloads for efficient batch execution

Reduces the time to onboard new projects and applications

The benefits of Kubernetes extend beyond the development, testing, and production environment to perform mission-critical tasks in large-scale businesses.

**What is Kubernetes?**

Kubernetes is also known as '**k8s'.** This word comes from the Greek language, which means **a pilot or helmsman.**

Kubernetes is an extensible, portable, and open-source platform designed by **Google in 2014**. It is mainly used to automate the deployment, **scaling**, and operations of the container-based applications across the cluster of nodes. It is also designed for managing the services of containerized apps using different methods which provide the scalability, predictability, and high availability.

It is actually an enhanced version of 'Borg' for managing the long-running processes and batch jobs. Nowadays, many cloud services offer a Kubernetes-based infrastructure on which it can be deployed as the platform-providing service. for example Azure has AKS (azure kubernetes service).

This technique or concept works with many container tools, like docker, and follows the client-server architecture.

**Key Objects of Kubernetes**

Following are the key objects which exist in the Kubernetes:

**Pod**

It is the smallest and simplest basic unit of the Kubernetes application. This object indicates the processes which are running in the cluster. it has application/container and volume

**Node**

A node is nothing but a single host, which is used to run the virtual or physical machines. A node in the Kubernetes cluster is also known as a minion.

**Service**

A service in a Kubernetes is a logical set of pods, which works together. With the help of services, users can easily manage load balancing configurations.

**ReplicaSet**

A ReplicaSet in the Kubernetes is used to identify the particular number of pod replicas are running at a given time. It replaces the replication controller because it is more powerful and allows a user to use the "set-based" label selector.

**Namespace**

Kubernetes supports various virtual clusters, which are known as namespaces. It is a way of dividing the cluster resources between two or more users.

**Features of Kubernetes**

Following are the essential features of Kubernetes:

**Pod:** It is a deployment unit in Kubernetes with a single Internet protocol address.

**Horizontal Scaling**: It is an important feature in the Kubernetes. This feature uses a HorizontalPodAutoscalar to automatically increase or decrease the number of pods in a deployment, replication controller, replica set, or stateful set on the basis of observed CPU utilization.

**Automatic Bin Packing**: Kubernetes helps the user to declare the maximum and minimum resources of computers for their containers.

**Service Discovery and load balancing:** Kubernetes assigns the IP addresses and a Name of DNS for a set of containers, and also balances the load across them.

**Automated rollouts and rollbacks:** Using the rollouts, Kubernetes distributes the changes and updates to an application or its configuration. If any problem occurs in the system, then this technique rollbacks those changes for you immediately.

**Persistent Storage:** Kubernetes provides an essential feature called 'persistent storage' for storing the data, which cannot be lost after the pod is killed or rescheduled. Kubernetes supports various storage systems for storing the data, such as **Google Compute Engine's Persistent Disks** (GCE PD) or **Amazon Elastic Block Storage** (EBS). It also provides the distributed file systems: NFS or GFS.

**Self-Healing:** This feature plays an important role in the concept of Kubernetes. Those containers which are failed during the execution process, Kubernetes restarts them automatically. And, those containers which do not reply to the user-defined health check, it stops them from working automatically.

**Architecture of Kubernetes**



The architecture of Kubernetes actually follows the client-server architecture. It consists of the following two main components:

**Master Node (Control Plane)**

**Slave/worker node**

**Master Node or Kubernetes Control Plane**

The master node in a Kubernetes architecture is used to manage the states of a cluster. It is actually an entry point for all types of administrative tasks. In the Kubernetes cluster, more than one master node is present for checking the fault tolerance.

Following are the four different components which exist in the Master node or Kubernetes Control plane:

API Server

Scheduler

Controller Manager

ETCD

**API Server**

The Kubernetes API server receives the REST commands which are sent by the user. After receiving, it validates the REST requests, process, and then executes them. After the execution of REST commands, the resulting state of a cluster is saved in 'etcd' as a distributed key-value store.

**Scheduler**

The scheduler in a master node schedules the tasks to the worker nodes. And, for every worker node, it is used to store the resource usage information.

In other words, it is a process that is responsible for assigning pods to the available worker nodes.

**Controller Manager**

The Controller manager is also known as a controller. It is a daemon that executes in the non-terminating control loops. The controllers in a master node perform a task and manage the state of the cluster. In the Kubernetes, the controller manager executes the various types of controllers for handling the nodes, endpoints, etc.

**ETCD**

It is an open-source, simple, distributed key-value storage which is used to store the cluster data. It is a part of a master node which is written in a GO programming language.

**Now, we have learned about the functioning and components of a master node; let's see what is the function of a slave/worker node and what are its components.**

Worker/Slave node

The Worker node in a Kubernetes is also known as minions. A worker node is a physical machine that executes the applications using pods. It contains all the essential services which allow a user to assign the resources to the scheduled containers.

Following are the different components which are presents in the Worker or slave node:

**Kubelet**

This component is an agent service that executes on each worker node in a cluster. It ensures that the pods and their containers are running smoothly. Every kubelet in each worker node communicates with the master node. It also starts, stops, and maintains the containers which are organized into pods directly by the master node.

**Kube-proxy**

It is a proxy service of Kubernetes, which is executed simply on each worker node in the cluster. The main aim of this component is request forwarding. Each node interacts with the Kubernetes services through Kube-proxy.

**Pods**

A pod is a combination of one or more containers which logically execute together on nodes. One worker node can easily execute multiple pods.