**Kubernetes Overview:**

Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications.

Kubernetes in an open source container management tool hosted by Cloud Native Computing Foundation (CNCF).

Kubernetes is a container management technology developed in Google lab to manage containerized applications in different kind of environments such as physical, virtual, and cloud infrastructure.

Features of Kubernetes:

Following are some of the important features of Kubernetes.

1. Containerized infrastructure

2. Application-centric management

3. Auto-scalable infrastructure

4. Environment consistency across development testing and production

5. Loosely coupled infrastructure, where each component can act as a separate unit

6. Higher density of resource utilization

7. Predictable infrastructure which is going to be created

8. Continues development, integration and deployment

## Why you need Kubernetes and what can it do

Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime.

It takes care of your scaling requirements, failover, deployment patterns, and more. For example, Kubernetes can easily manage a canary deployment for your system.

Kubernetes provides you with:

* **Service discovery and load balancing**  
  Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable.
* **Storage orchestration**  
  Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.
* **Automated rollouts and rollbacks**  
  You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.
* **Automatic bin packing**  
  Kubernetes allows you to specify how much CPU and memory (RAM) each container needs. When containers have resource requests specified, Kubernetes can make better decisions to manage the resources for containers.
* **Self-healing**  
  Kubernetes restarts containers that fail, replaces containers, kills containers that don’t respond to your user-defined health check, and doesn’t advertise them to clients until they are ready to serve.
* **Secret and configuration management**  
  Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.

# **Kubernetes Components**

Master components provide the cluster’s control plane. Master components make global decisions about the cluster (for example, scheduling), and they detect and respond to cluster events (for example, starting up a new [pod](https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/) when a deployment’s **replicas** field is unsatisfied).

Master components can be run on any machine in the cluster. However, for simplicity, set up scripts typically start all master components on the same machine, and do not run user containers on this machine.

1. Kube-apiserver
2. ectd
3. Kube-scheduler
4. Kube-controller-manager

Kube-apiserver: Component on the master that exposes the Kubernetes API. It is the front-end for the Kubernetes control plane.

ectd: Consistent and highly-available key value store used as Kubernetes’ backing store for all cluster data.

Kube-scheduler: Component on the master that watches newly created pods that have no node assigned, and selects a node for them to run on.

Kube-controller-manager: Component on the master that runs [controllers](https://kubernetes.io/docs/admin/kube-controller-manager/).

These controllers include:

* Node Controller: Responsible for noticing and responding when nodes go down.
* Replication Controller: Responsible for maintaining the correct number of pods for every replication controller object in the system.
* Endpoints Controller: Populates the Endpoints object (that is, joins Services & Pods).
* Service Account & Token Controllers: Create default accounts and API access tokens for new namespaces.

**The Node components are as follows:**

Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment.

1. Kubelet: An agent that runs on each node in the cluster. It makes sure that containers are running in a pod.
2. Kube-proxy: [kube-proxy](https://kubernetes.io/docs/reference/command-line-tools-reference/kube-proxy/) is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes [Service](https://kubernetes.io/docs/concepts/services-networking/service/" \t "_blank)concept. kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.
3. Container Runtime: The container runtime is the software that is responsible for running containers.

**## Kubernetes Alternatives & Competitors**

1. Amazon ECS

2. Docker Platform(Docker Swarm)

3. Apache Mesos

4. Helios

5. Marathon

6. Centurion

7. Cloudify

https://kubernetes.io/docs/tutorials/hello-minikube/-------- link for deployment of Application

Link for kubernetes questions (Refer to the questions)

<https://www.mirantis.com/blog/multi-container-pods-and-container-communication-in-kubernetes/>

**CI/CD in kubernetes**

1. Automation allows new features to be tested, deployed and delivered to your customers much more quickly.

The ability to deliver more, faster, not only gives you a competitive advantage but it also leaves more time

for innovation within your organization; and since adjustments and updates are rolled out rapidly, your customers

will also be happier.

kubernets allows you to spin up a number of container instances and manage them for scaling and fault tolerance.

It also handles a wide range of management activities that would otherwise require separate solutions or custom code,

including request routing, container discovery, health checking, and rolling updates.

**## Why Automate your CI/CD Pipeline?**

Manually updating your app in Kubernetes and why that’s a painful process.

Achieving Continuous Integration/Continuous Deployment (CI/CD) - the components and features you need.

Approaches to continuous delivery and an overview of CI/CD tools available to build your own automated pipeline.

## Kubernetes Continuous Deploy

**##** **Achieving Continuous Delivery in Kubernetes**

Standing up a Kubernetes cluster today is a rather straightforward operation, but what isn’t so simple is getting your

updates or new features rolled out to your running cluster. There are a number of options available that can help you automate

a continuous delivery pipeline to Kubernetes without you and your team having to manually deploying changes.

These are the components that make up a Continuous Delivery (CD) pipeline:

**Version Control System** – a source code repository where changes and updates are pushed

**CI system** – an integration and test system that may also build the Docker image

**Docker Registry** – the image registry that stores your Docker images

**Kubernetes Cluster** – set up with a few clicks in GKE or using one of the available installers

**## Approaches to Continuous Delivery to Kubernetes**

Even though there are a number of tools available, CI/CD vendors are still in the process of defining how a CI/CD Kubernetes pipeline

can and should look like for engineers who create cloud native applications.

The key to Continuous Delivery is to coordinate and automate these pieces to increase velocity and make Kubernetes deployments

seamless.

https://plugins.jenkins.io/kubernetes-cd

https://cloud.google.com/solutions/continuous-delivery-jenkins-kubernetes-engine

https://www.weave.works/technologies/ci-cd-for-kubernetes/ -------- CI/CD for Kubernetes

**There are two types of volumes**

a. Ephimeral (Short lived)

b. Durable

1. emptyDir

An emptyDir volume is first created when a Pod is assigned to a Node, and exists as long as that Pod is running on that node.

As the name says, it is initially empty. Containers in the Pod can all read and write the same files in the emptyDir volume,

though that volume can be mounted at the same or different paths in each Container. When a Pod is removed from a node for

any reason, the data in the emptyDir is deleted forever.

2. Hostpath

A hostPath volume mounts a file or directory from the host node’s filesystem into your Pod. This is not something that most

Pods will need, but it offers a powerful escape hatch for some applications. For Ex:

Running a Container that needs access to Docker internals; use a hostPath of /var/lib/docker

3. AwsElastic Block Store

2. GCEPersistant Disk

3. Azure Disk

4. Azure File

5. ISCSI

6. NFS

7. Cinder

8. VsphereVolume

Statefull applications are run in containers. Statefull apps data can be from webservers or applications.

Pods are ephemeral.

Volumes are required to store data. Volumes are associated with lifecycle of pods.

Kubernetes support multiple number of volumes and can use multiple volumes at one time.

Emptydir volume is created when pod is created. It is also used as temporary space to share data between the containers.

It is deleted once pod is deleted.

Hostpath is used to mount the file or directory from the host's file system to the pod.

It can be used for containers which needs access to docker internals--- /var/lib/docker

**Secret** : Kubernetes secret objects let you store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys.

You can store secrets in the Kubernetes API and mount them as files for use by Pods without coupling to Kubernetes directly.

secret volumes are backed by tmpfs (a RAM-backed filesystem) so they are never written to non-volatile storage.

Type of storage How long does it last?

Container filesystem Container lifetime

Volume Pod lifetime

Persistent volume Cluster lifetime

Commands to create volumes:

aws ec2 create-volume --availability-zone=eu-west-1a --size=10 --volume-type=gp2

gcloud compute disks create --size=500GB --zone=us-central1-a my-data-disk

**Networking in Kubernetes:**

Kubernetes does not provide any default network implementation, rather it only defines the model and leaves to other tools

to implement it.

All containers(Pod) will be assigned one ip address in this overlay network, they communicate with each other by calling each

other’s ip address directly.

Flannel’s idea is simple: create another flat network which runs above the host network, this is the so-called overlay network.

1. **To run Kubernetes we need to install minikube on the system.**

# minikube version: Used to check the version of minikube installed on machine

# minikube start: Used to start minikube

# kubectl cluster-info : To check the cluster details

# kubectl get nodes : To check node details. Shows all nodes which will be used to host application

1. Once application instances are created, the Kubernetes Deployment Controller continuously monitors those instances. If the Node hosting an instance goes down or is deleted, the Deployment controller replaces the instance with an instance on another Node in the cluster. **This provides a self-healing mechanism to address machine failure or maintenance.**
2. Kubectl uses the Kubernetes API to interact with the cluster. Kubectl command is needed to create Deployments that run your applications on a Kubernetes cluster.
3. When we create a Deployment, we need to specify the container image for the application and the number of replicas that we want to run.
4. Kubectl version: Used to check the client and server version.
5. Kubectl run command I used to create deployment. The **run** command is used to creates a new deployment. If you want to run the app on a specific port so we add the **--port** parameter:

# kubectl get deployment: used to create a deployment on the node

1. A Pod is a Kubernetes abstraction that represents a group of one or more application containers (such as Docker or rkt), and some shared resources for those containers.
2. Each Pod is tied to the Node where it is scheduled, and remains there until termination (according to restart policy) or deletion. In case of a Node failure, identical Pods are scheduled on other available nodes in cluster.
3. A Pod always runs on a **Node**. A Node is a worker machine in Kubernetes and may be either a virtual or a physical machine, depending on the cluster. Each Node is managed by the Master. A Node can have multiple pods, and the Kubernetes master automatically handles scheduling the pods across the Nodes in the cluster.

Every Kubernetes Node runs at least:

* Kubelet, a process responsible for communication between the Kubernetes Master and the Node; it manages the Pods and the containers running on a machine.
* A container runtime (like Docker, rkt) responsible for pulling the container image from a registry, unpacking the container, and running the application.

1. Troubleshooting with kubectl:

* **kubectl get pod** - list resources (pods)
* **kubectl describe** **pods**- show detailed information about a resource (pods)
* **kubectl logs** - print the logs from a container in a pod
* **kubectl exec** - execute a command on a container in a pod

1. kubectl get namespace: Namespaces are intended for use in environments with many users spread across multiple teams, or projects. Namespaces provide a scope for names. Names of resources need to be unique within a namespace, but not across namespaces.

Links:

<https://kubernetes.io/docs/tutorials/hello-minikube/>

<https://kubernetes.io/docs/tutorials/kubernetes-basics/>

<https://www.mulesoft.com/resources/api/what-is-an-api> ----API Link

<https://kubernetes.io/docs/tutorials/kubernetes-basics/create-cluster/cluster-interactive/--> Online tutorial link for Kubernetes.

<https://courses.edx.org/courses/course-v1:LinuxFoundationX+LFS158x+2T2019/courseware/e406f02239a7439593d260b658962fef/c77f2d1b11234afb9886680612041ab2/?child=first> ----- Linux Foundation Course link. Covers all topics precisely

**What are Microservices**

**Microservices** are lightweight applications written in various modern programming languages, with specific dependencies, libraries and environmental requirements. To ensure that an application has everything it needs to run successfully it is packaged together with itsdependencies.

container orchestration tools and services available today

**Amazon Elastic Container Service**

**Azure Container Instances**

**Azure Service Fabric**

**Kubernetes**

**Marathon**

**Nomad**

**Docker Swarm**