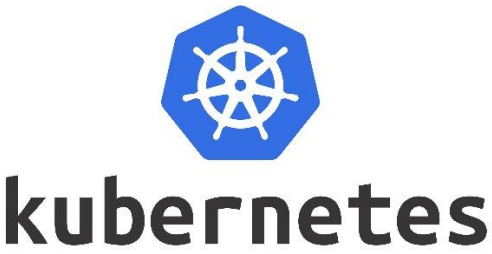

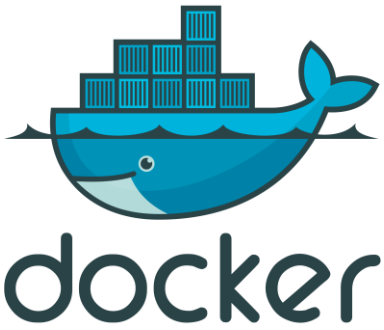

What is Kubernetes?

Kubernetes is like a smart manager for computer programs. Imagine you have lots of small workers (like virtual machines or containers) that need to work together on a project. Kubernetes makes sure these workers do their jobs correctly, keeps track of their tasks, and helps them communicate with each other. It ensures that your programs run smoothly, even if some workers fail or new ones are added. In short, Kubernetes helps manage and organize your software

Difference between Docker and Kubernetes:

Docker: Imagine you have small, self-contained boxes for your apps. These boxes have everything the apps need to run. Docker is like the tool that makes these boxes and ensures each app has its own safe space to run.

Kubernetes: Now, think of a super-smart manager who takes care of all these boxes. This manager makes sure the boxes are in the right place, creates more boxes when needed, and replaces any broken ones. It also spreads the work evenly among the boxes, so everything runs smoothly. That's Kubernetes – the manager for your app boxes.

Features of Kubernetes:

Certainly, here are some key features of Kubernetes:

1. **Container Orchestration**: Kubernetes automates the deployment, scaling, and management of containerized applications, allowing you to focus on your code rather than managing infrastructure.
2. **Automated Load Balancing**: Kubernetes balances network traffic across your containers, ensuring even distribution and efficient use of resources.
3. **Auto Scaling**: Kubernetes can automatically adjust the number of running containers based on demand, helping you efficiently handle varying levels of traffic.
4. **Self-Healing**: If a container fails or becomes unresponsive, Kubernetes automatically restarts it or replaces it with a new one to maintain application availability.
5. **Rolling Updates and Rollbacks**: Kubernetes allows you to update your application without downtime by gradually replacing old containers with new ones. If something goes wrong, you can easily roll back to the previous version.
6. **Service Discovery**: Kubernetes provides an internal DNS system to automatically manage and discover the IP addresses of containers and services within your cluster.
7. **Storage Orchestration**: Kubernetes can manage storage systems, automatically attaching volumes to containers as needed. This helps with data persistence and sharing.
8. **Configuration Management**: Kubernetes lets you define, manage, and update application configurations and environment variables separately from your application code.
9. **Secrets Management**: Kubernetes provides a secure way to manage sensitive information, such as passwords and API keys, by storing and distributing secrets to containers.
10. **Multi-Environment Deployment**: Kubernetes supports deploying applications consistently across various environments, from development to production, making it easier to maintain consistency.
11. **Horizontal Pod Autoscaling**: In addition to scaling the number of replicas, Kubernetes can automatically adjust the number of pods within a deployment based on CPU or custom metrics.
12. **Namespace Isolation**: Kubernetes allows you to create multiple virtual clusters within the same physical cluster, providing isolation and separation for different projects or teams.
13. **Batch Processing**: Kubernetes supports batch processing workloads, allowing you to run tasks at scheduled intervals or based on specific conditions.
14. **Resource Utilization Monitoring**: Kubernetes provides tools to monitor and visualize the resource usage of your containers, helping you optimize performance.

15. **Extensibility**: Kubernetes is highly extensible and can be customized using plugins and APIs to integrate with other tools and services.

These features collectively make Kubernetes a powerful platform for deploying, managing, and scaling containerized applications in a cloud-native environment.

History:

Kubernetes started at Google, became open-source in 2014, and quickly grew into a leading platform for managing containers. It forms the basis for modern cloud applications, allowing easy deployment, scaling, and management of software. As part of the Cloud Native Computing Foundation, Kubernetes has a thriving ecosystem and continues to evolve, making complex tasks simpler for developers and operations teams.

Kubernetes Cloud Providers:

Certainly, here are some popular cloud-based Kubernetes services offered by major cloud providers:

1. **Google Kubernetes Engine (GKE)**: Managed Kubernetes service from Google Cloud Platform (GCP). It provides automated updates, scaling, and management of Kubernetes clusters.
2. **Amazon Elastic Kubernetes Service (EKS)**: Managed Kubernetes service provided by Amazon Web Services (AWS). It simplifies the deployment, management, and scaling of Kubernetes clusters.
3. **Microsoft Azure Kubernetes Service (AKS)**: Managed Kubernetes service on Microsoft Azure. It offers automated Kubernetes cluster management and seamless integration with Azure services.

Difference in between Kubernetes and Docker Swarm:

Kubernetes:

- More complex, flexible, and powerful.

- Supports advanced scaling, multi-cluster management, and complex networking.
- Large ecosystem, mature, and widely used for production.
- Declarative configuration approach.
- Well-suited for large-scale, diverse applications.

Docker Swarm:

- Simpler and closely integrated with Docker.
- Offers basic scaling and networking.
- Smaller community and ecosystem.
- Supports both declarative and imperative configuration.
- Good for smaller projects or teams familiar with Docker.

Kubernetes is a powerful open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. It consists of various components that work together to provide these functionalities. Here are the core components of Kubernetes and a brief overview of how they work:

1. **Master Components:** The master components manage the overall Kubernetes cluster and make global decisions about the cluster (e.g., scheduling).
    - **API Server:** The API server is the central control point for the Kubernetes cluster. It exposes the Kubernetes API, which is used by both internal components and external users to interact with the cluster. All other components communicate with the API server to read and modify the desired state of the system.
    - **Controller Manager:** The controller manager is responsible for maintaining the desired state of the system. It includes various controllers that monitor and regulate the state of different resources, ensuring that the current state matches the desired state.
    - **Scheduler:** The scheduler assigns newly created pods to available nodes based on various factors like resource requirements, quality of service, affinity, and anti-affinity rules. It aims to optimize resource utilization and balance the workload across the cluster.
    - **etcd:** etcd is a distributed key-value store that stores the configuration data of the cluster. It acts as the cluster's "source of truth" and is used by the other components to store and retrieve configuration data.
2. **Node Components:** Node components run on each worker node and manage the containers running on that node.

- **Kubelet:** The kubelet is responsible for ensuring that containers are running in a Pod. It communicates with the control plane to receive the desired state of the pods and takes actions to maintain the actual state.
- **Kube Proxy:** Kube Proxy is a network proxy that maintains network rules on nodes. It manages network connectivity between pods and services by setting up routes, load balancing, and handling network traffic.
- **Container Runtime:** The container runtime is responsible for running containers within pods. Popular runtimes include Docker, containerd, and CRI-O. The runtime takes care of pulling container images, creating containers, and managing their lifecycle.

3. **Pods:** Pods are the smallest deployable units in Kubernetes and encapsulate one or more containers. They share the same network namespace, storage, and IP address.

These components work together to manage the lifecycle of containerized applications, ensure high availability, and abstract away the complexities of infrastructure management, enabling developers to focus on writing and deploying applications.

Minikube Installation:

Take ubuntu t3 medium instance

Switch to root user and run below command

```
apt update && apt -y install docker.io
```

Switch to ubuntu user and run below commands

```
wget
https://storage.googleapis.com/minikube/releases/latest/minikube-
linux-amd64
sudo cp minikube-linux-amd64 /usr/local/bin/minikube
sudo chmod 755 /usr/local/bin/minikube
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
sudo install -o root -m  0755 kubectl /usr/local/bin/kubectl
kubectl version
sudo usermod -aG docker $USER && newgrp docker
minikube start –driver=docker
minikube status
```

Note: Make sure you terminate the instance once done with your practice

---

Here are those basic Minikube commands explained in simpler terms:

1. **Start Minikube Cluster**:
   Start your local Kubernetes cluster using Minikube. It sets up a mini-version of Kubernetes on your computer.
   ```
   minikube start
   ```
2. **Check Cluster Status**:
   Check if your Minikube cluster is running and see its status.
   ```
   minikube status
   ```
3. **Stop Minikube Cluster**:
   Stop the Minikube cluster if you're not using it. It's like turning off your Kubernetes playground.
   ```
   minikube stop
   ```
4. **Delete Minikube Cluster**:
   Completely remove your Minikube cluster. It's like uninstalling your Kubernetes playground.
   ```
   minikube delete
   ```
5. **List Available Kubernetes Contexts**:
   See a list of your Kubernetes environments, including Minikube.
   ```
   kubectl config get-contexts
   ```

6. **Set the Current Context to Minikube**:
   Tell your computer to use the Minikube cluster as the active Kubernetes environment.
   ```
   kubectl config use-context minikube
   ```
7. **Run a Sample Application**:
   Deploy a simple test application, like a web server, into your Minikube cluster.
   ```
   kubectl create deployment nginx --image=nginx
   ```

8. **Expose the Application**:
   Make your test application accessible from outside the cluster. Think of it as opening a door to the web server.
   ```
   kubectl expose deployment nginx --port=80 --type=NodePort
   ```
9. **Access the Application**:
   Get the web address to view your test application in a web browser.
   ```
   minikube service nginx
   ```

Nginx is like a traffic cop for websites. It makes sure web pages get to you quickly and safely when you visit a site. It also helps manage the flow of internet traffic to keep things running smoothly.

Kubernetes deployment on AWS EC2 Instance

```
sudo su
yum update -y
yum install docker -y
systemctl start docker
yum repolist
```


You can find below steps from Kubernetes official website

```
# This overwrites any existing configuration in /etc/yum.repos.d/kubernetes.repo
cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-\$basearch
enabled=1
gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
exclude=kubelet kubeadm kubectl
EOF


# Set SELinux in permissive mode (effectively disabling it)
sudo setenforce 0
sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
```

sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
sudo systemctl enable --now kubelet

Run the below command on only master
kubeadm init

kubectl get nodes


```
apiVersion: batch/v1
kind: Job
metadata:
  name: testjob
spec:
  template:
    metadata:
      name: testjob
    spec:
      containers:
      - name: counter
        image: centos:7
        command: ["bin/bash", "-c", "echo Cloud Nation; sleep 5"]
      restartPolicy: Never
--------------
apiVersion: batch/v1
```

```
kind: Job
metadata:
  name: testjob
spec:
  parallelism: 5                    # Runs for pods in parallel
  activeDeadlineSeconds: 10  # Timesout after 30 sec
  template:
    metadata:
      name: testjob
    spec:
      containers:
      - name: counter
        image: centos:7
        command: ["bin/bash", "-c", "echo Cloud Nation; sleep 20"]
      restartPolicy: Never
```

```
kubectl apply -f job.yml
kubectl get pods
kubectl delete -f job.yml
watch !!
```