Kubernetes:Introduction to Container Orchestration

Kubernetes is an open-source Container Management tool that automates container deployment, container scaling, descaling, and container load balancing (also called a container orchestration tool). It is written in Golang and has a vast community because it was first developed by Google and later donated to CNCF (Cloud Native Computing Foundation).

Kubernetes is an open-source platform that manages Docker containers in the form of a cluster. Along with the automated deployment and scaling of containers, it provides healing by automatically restarting failed containers and rescheduling them when their hosts die. This capability improves the application’s availability.

Features of Kubernetes

1**. Automated Scheduling**– Kubernetes provides an advanced scheduler to launch containers on cluster nodes. It performs resource optimization.

2. **Self-Healing Capabilities**– It provides rescheduling, replacing, and restarting the containers which are dead.

3. **Automated Rollouts and Rollbacks**– It supports rollouts and rollbacks for the desired state of the containerized application.

4. **Horizontal Scaling and Load Balancing**– Kubernetes can scale up and scale down the application as per the requirements.

5. **Resource Utilization**– Kubernetes provides resource utilization monitoring and optimization, ensuring containers are using their resources efficiently.

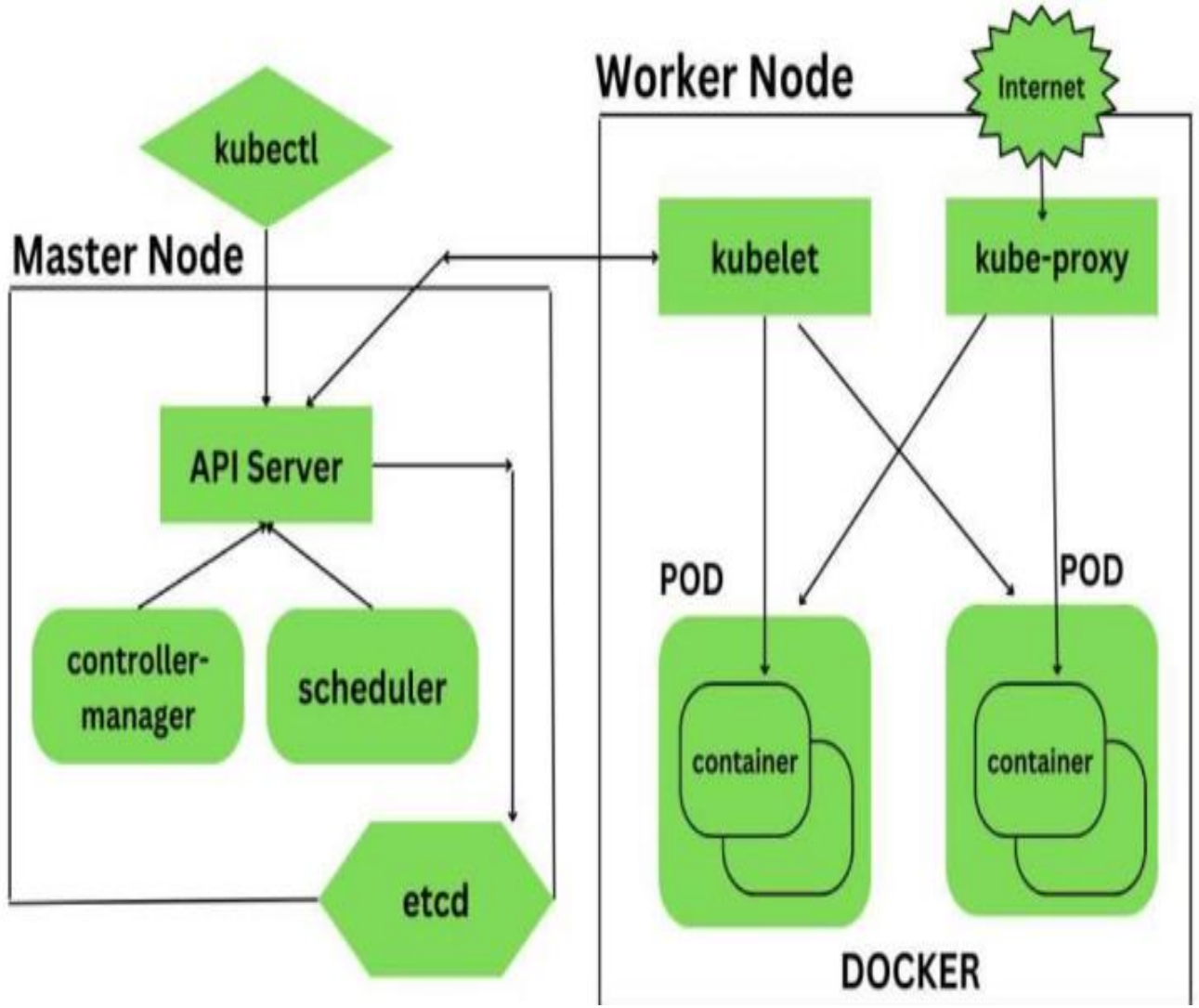
6. **Support for multiple clouds and hybrid clouds**– Kubernetes can be deployed on different cloud platforms and run containerized applications across multiple clouds.

7. Extensibility– Kubernetes is very extensible and can be extended with custom plugins and controllers.

8. **Community Support-** Kubernetes has a large and active community with frequent updates, bug fixes, and new features being added

Architecture of Kubernetes

- Kubernetes follows the client-server architecture where we have the master installed on one machine and the node on separate Linux machines. It follows the master-slave model, which uses a master to manage Docker containers across multiple Kubernetes
- nodes. A master and its controlled nodes(worker nodes) constitute a “Kubernetes cluster”. A developer can deploy an application in the docker containers with the assistance of the Kubernetes master.



Kubernetes- Master Node Components

This is the entry point of all administrative tasks

API Server– The API server is the entry point for all the REST commands used to control the cluster. All the administrative tasks are done by the API server within the master node. We can interact with these APIs using a tool called **kubectl**.

Scheduler– It is a service in the master responsible for distributing the workload.

Controller Manager– Also known as controllers. It is a daemon that runs in a non-terminating loop and is responsible for collecting and sending information to the API server. It regulates the Kubernetes cluster by performing lifestyle functions such as namespace creation and lifecycle event garbage collections, terminated pod garbage collection, cascading deleted garbage collection, node garbage collection, and many more.

etcd– It is a distributed key-value lightweight database. In Kubernetes, it is a central database for storing the current cluster state at any point in time and is also used to store the configuration details such as subnets, config maps, etc

Kubernetes- Worker Node Components

Kubernetes Worker node contains all the necessary services to manage the networking between the containers, communicate with the master node, and assign resources to the containers scheduled.

a**.) Kubelet–** It is a primary node agent which communicates with the master node and executes on each worker node inside the cluster. It gets the pod specifications through the API server and executes the container associated with the pods and ensures that the containers described in the pods are running and healthy.

b.) **Kube-Proxy–** It is the core networking component inside the Kubernetes cluster. It is responsible for maintaining the entire network configuration. Kube-Proxy maintains the distributed network across all the nodes, pods, and containers and exposes the services across the outside world. It acts as a network proxy and load balancer for a service on a single worker node and manages the network routing for TCP and UDP packets.

c**.) Pods–** A pod is a group of containers that are deployed together on the same host. With the help of pods, we can deploy multiple dependent containers together so it acts as a wrapper around these containers so we can interact and manage these containers primarily through pods.

d.) **Docker–** Docker is the containerization platform that is used to package your application and all its dependencies together in the form of containers to make sure that your application works seamlessly in any environment which can be development or test or production. Docker is a tool designed to make it easier to create, deploy, and run applications by using containers.

Kubernetes Namespaces

Kubernetes Namespace is a mechanism that enables you to organize resources. It is like a virtual cluster inside the cluster. A namespace isolates the resources from the resources of other namespaces. For example, You need to have different names for deployments/services in a namespace but you can have the same name for deployment in two different namespaces.

By default, Kubernetes has 4 namespaces:

```
ishan301@G3-3500:~$ kubectl get namespaces
NAME              STATUS   AGE
default           Active   3h11m
kube-node-lease   Active   3h11m
kube-public       Active   3h11m
kube-system       Active   3h11m
```

- **kube-system:** System processes like Master and kubectl processes are deployed in this namespace; thus, it is advised NOT TO CREATE OR MODIFY ANYTHING namespace.
- **kube-public:** This namespace contains publicly accessible data like a configMap containing cluster information.
- **kube-node-lease:** This namespace is the heartbeat of nodes. Each node has its associated lease object. It determines the availability of a node.
- **default:** This is the namespace that you use to create your resources by default.

Kubernetes- Worker Node Components

kubectl create namespace your-namespace

Creating Component in a Namespace:To create a component in a namespace you can either give the –namespace flag or specify the namespace in the configuration file.

Method 1: Using –namespace flag

$ kubectl apply -f your_config.yaml --namespace=your-namespace

then you can check resources in your namespace using **kubectl get** and specify namespace using **-n**

```
ishan301@G3-3500:~/play around$ kubectl apply -f nginx.yaml --namespace=gfg
deployment.apps/nginx created
ishan301@G3-3500:~/play around$ kubectl get deployment -n gfg
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   1/1     1            1           36s
ishan301@G3-3500:~/play around$
```

Method 2: Adding namespace in the configuration file

Instead of specifying a namespace using the –namespace flag you can specify your namespace initially in your config file only.

nginx.yaml

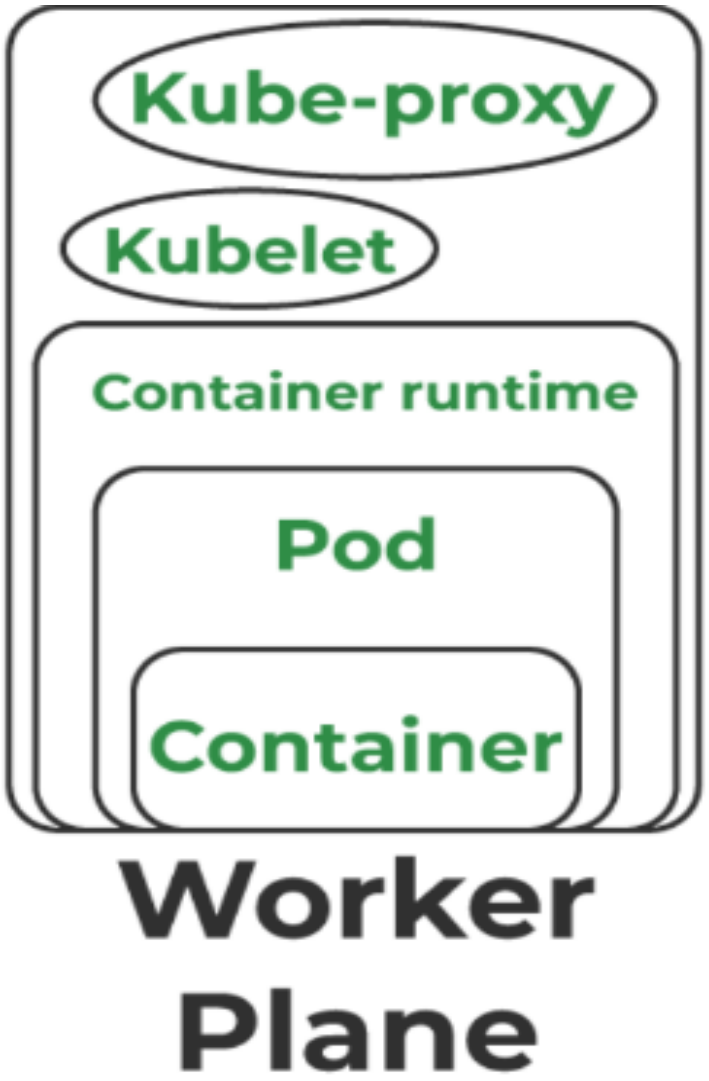
```yaml
apiVersion: apps/v1
kind: Deployment

metadata:
  name: nginx
  namespace: gfg
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        resources:
          limits:
            memory: "128Mi"
            cpu: "500m"
        ports:
        - containerPort: 800
```

and then use the command.

$ kubectl apply -f your_config_file.yaml

Kubernetes – Node

- Nodes are the Worker machines where the actual work happens. Each node has the services required to execute Pods and is controlled by the Control Plane. Each Node can have multiple pods and pods have containers running inside them. There are 3 processes in every Node that are used to Schedule and manage those pods.
- **Container runtime**: A container runtime is needed to run the application containers running on pods inside a pod. Example-> Docker
- **kubelet**: kubelet interacts with both the container runtime as well as the Node. It is the process responsible for starting a pod with a container inside.
- **kube-proxy:** It is the process responsible for forwarding the request from Services to the pods. It has intelligent logic to forward the request to the right pod in the worker node.



Node Status:

To view a Node's status and other information, use kubectl:

- **$ kubectl describe node <node-name>**

Kubernetes – PODS

A pod is the smallest unit that exists in Kubernetes. It is similar to that of tokens in C or C++ language. A specific pod can have one or more applications. The nature of Pods is ephemeral this means that in any case if a pod fails then Kubernetes can and will automatically create a new replica/ duplicate of the said pod and continue the operation.

Types of Pods

- Single Container Pod
- Multi Container Pod
- Static Pod

Create Pod

$ kubectl create -f [FILENAME]

Delete Pod

$ kubectl delete -f FILENAME

Get The Pod

- To see the no.of pods available in the particular namespace can be seen using the below command.

$Kubectl get pod <name> --namespace

ConfigMap in Kubernetes

ConfigMaps can be used to store configuration data as key-value pairs or as files. You can then use environment variables or volume mounts to make this data available to your applications running in Kubernetes.

To create a ConfigMap from a literal value, use the following command:

kubectl create configmap configmap-1 --from-literal=name=first-configmap

To create a ConfigMap from multiple literal values, use the following command:

kubectl create configmap configmap-2 --from-literal=name=second-configmap --from-literal=color=blue

To create a ConfigMap from a file, use the following command:

kubectl create configmap configmap-3 --from-file=data-file

To see the pods running in your Kubernetes cluster, use the following command:

kubectl get pods

To see the ConfigMaps created in your Kubernetes cluster, use the following command:

kubectl get cm

To see the environment variables set in a running pod, use the following command:

kubectl exec -it <pod-name> -- printenv

To go inside a running pod, use the following command:

kubectl exec -it <pod-name> -- bash

Kubernetes Secrets: How to Create, Use, and Manage Secrets in Kubernetes

Kubernetes secrets are an important aspect of managing Kubernetes applications. They allow you to store sensitive data securely and manage access to that data.

To create different types of secrets

Generic Secret

kubectl create secret generic db-secret --from-literal=username=dbuser --from-literal=password=Y4nys7f11

Docker-registry Secret

kubectl create secret docker-registry docker-secret --docker-email=example@gmail.com --docker-username=dev --docker-password=pass1234 --docker-server=my-registry.example:5000

TLS Secret

kubectl create secret tls my-tls-secret --cert=/root/data/serverca.crt --key=/root/data/servercakey.pem

Kubernetes Secrets: How to Create, Use, and Manage Secrets in Kubernetes

To apply yaml files

kubectl apply -f <file-name.yaml>

To check running pods

kubectl get pods

To check created secrets

kubectl get secret

To describe a secret

kubectl describe secret <secret-name>

To check environment variables in running pod

kubectl exec -it <pod-name> -- printenv

To go inside running pod

kubectl exec -it <pod-name> -- bash

To extract pod's yaml file

kubectl get pod <pod-name> -o yaml

To decode the encrypted data

echo "<data>" | base64 -d

Lab1

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-nginx-service
spec:
  selector:
    app: nginx
  ports:
  - protocol: TCP
    port: 80
    nodePort: 30001
  type: NodePort
```

Services.yaml

$Kubectl get all

$ Kuberctl apply –f services.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apache-deployment
  labels:
    app: nginx
spec:
  replicas: 10
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: httpd
        image: httpd
```

deployment.yaml

$ Kuberctl apply –f deployment.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-nginx-pod-2
  labels:
    app: nginx
spec:
  containers:
  - name: nginx-container
    image: nginx:1.17.0
```

pod.yaml

$ Kuberctl apply –f pod.yaml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: my-nginx-rs
spec:
  replicas: 5
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
```

replicaset.yaml

$ Kuberctl apply –f replicaset.yaml

Lab2:ConfigMap

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: configmap-demo-1
spec:
  containers:
  - name: demo-container
    image: nginx
    envFrom:
    - configMapRef:
        name: configmap-1
```

configMap1.yaml

$kubectl create configmap configmap-1 --from-literal=name=first-configmap

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: configmap-demo-2
spec:
  containers:
  - name: demo-container
    image: nginx
    env:
    - name: COLOR
      valueFrom:
        configMapKeyRef:
          name: configmap-2
          key: color
```

configMap2.yaml

$ kubectl create configmap configmap-2 --from-literal=name=second-configmap --from-literal=color=blue

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: configmap-demo-3
spec:
  containers:
  - name: demo-container
    image: nginx
    volumeMounts:
      - name: config
        mountPath: /etc/config
  volumes:
  - name: config
    configMap:
      name: configmap-3
```

ConfigMap3.yaml

$ kubectl create configmap configmap-3 --from-file=data-file

Lab3:Secrets

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secret-demo-1
spec:
  containers:
  - name: demo-container
    image: nginx
    env:
    - name: Username
      valueFrom:
        secretKeyRef:
          name: db-secret
          key: username
```

secret1.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secret-demo-2
spec:
  containers:
  - name: demo-container
    image: nginx
    envFrom:
    - secretRef:
        name: docker-secret
```

Secret2.yaml