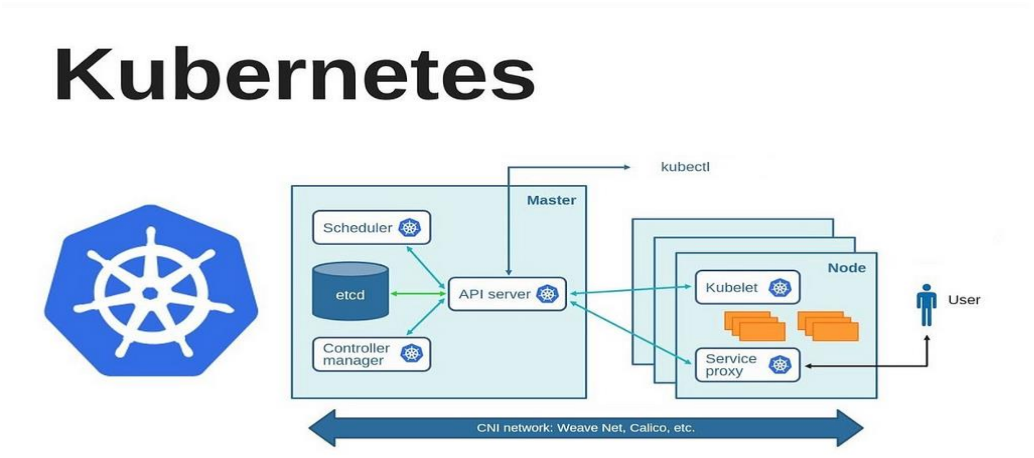
Kubernetes is an open-source, cloud-native platform that allows you to deploy, scale, and manage your applications across multiple nodes and clusters.

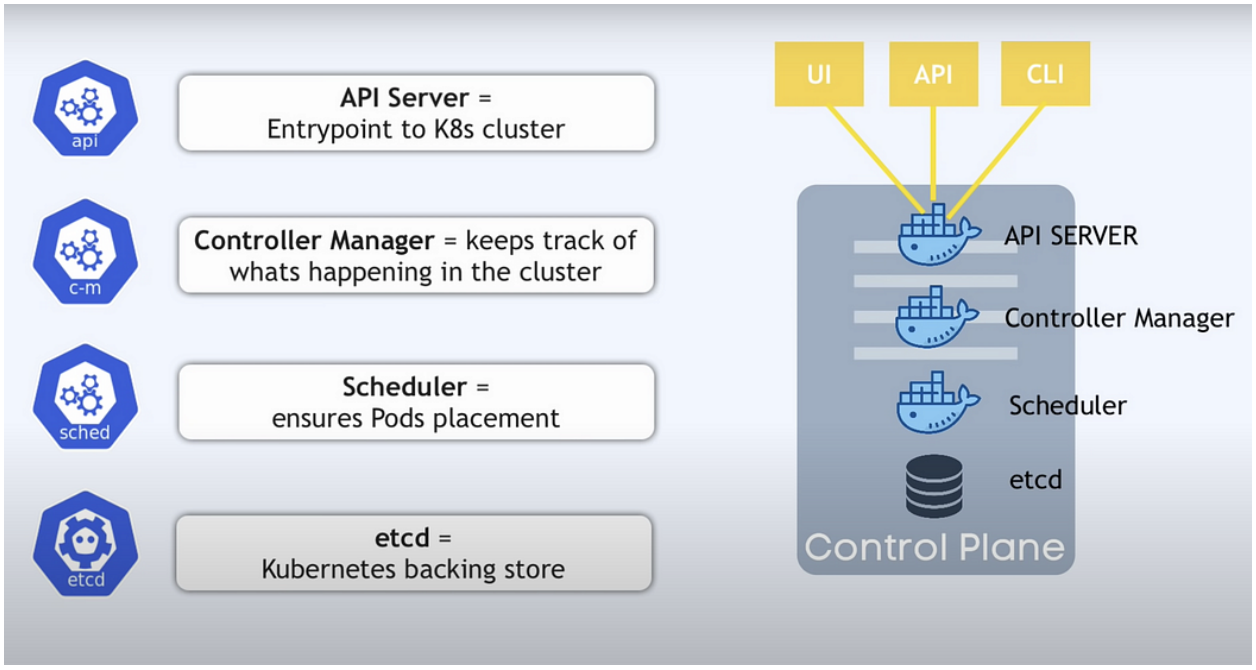
Benefits of using Kubernetes

Kubernetes provides many benefits for developers and operators who want to run their applications in containers. Some of the benefits are:

- **High availability**: Kubernetes ensures that your applications are always up and running by automatically restarting failed containers, rescheduling them to different nodes, or scaling them to meet demand.
- **Load balancing**: Kubernetes distributes the traffic among your containers by exposing them as services with their own IP addresses and DNS names. It also supports service discovery and routing mechanisms to connect your services with each other.
- **Scalability**: Kubernetes allows you to easily scale your applications up or down by changing the number of replicas or using horizontal pod autoscaling. You can also scale your cluster by adding or removing nodes as needed.
- **Portability**: Kubernetes enables you to run your applications on any platform that supports Kubernetes, such as public clouds, private clouds, hybrid clouds, or bare metal servers. You can also migrate your applications across different environments without changing your code or configuration.
- **Automation**: Kubernetes automates many tasks that are otherwise tedious or error-prone, such as deployment, rollback, configuration management, secret management, service discovery, resource allocation, health monitoring, etc.
- **Extensibility**: Kubernetes is designed to be modular and pluggable, so you can customize it to suit your needs. You can use various tools and plugins that integrate with Kubernetes, such as Helm, Istio, Prometheus, etc. You can also extend Kubernetes by adding your own custom resources and controllers.

Architecture of Kubernetes





API Server =
Entrypoint to K8s cluster

Controller Manager = keeps track of whats happening in the cluster

Scheduler =
ensures Pods placement

etcd =
Kubernetes backing store

Control plane

The control plane is the component of Kubernetes that controls and manages the state and configuration of the cluster. It consists of several components that communicate with each other through the API server. The control plane components are:

- API server: The central hub that exposes the Kubernetes API and processes all requests from clients and other components.
- etcd: A distributed key-value store that stores the cluster data in a consistent and reliable way.

- scheduler: Responsible for assigning pods to nodes based on various factors such as resource requirements, affinity rules, etc.
- controller manager: Runs various controllers that handle different aspects of the cluster such as node lifecycle, replication, service endpoints, etc.
- cloud controller manager: Interacts with the underlying cloud provider to manage cloud-specific resources such as load balancers, storage volumes, etc.

The control plane components can run on a single node or multiple nodes for high availability and fault tolerance. The control plane node is also called the master node.

Difference between kubectl **and** kubelet

kubectl and kubelet are two different tools that are used to interact with Kubernetes. The main difference between them are:

- kubectl is a command-line interface (CLI) tool that allows you to communicate with the API server and perform various operations on the cluster, such as creating, updating, deleting, or inspecting resources. You can use kubectl on any machine that can access the API server, such as your local machine or a remote server.
- kubelet is an agent that runs on each node in the cluster and communicates with the API server to register the node, report its status, and execute pod operations. You can use kubelet to manage the pods and containers on a specific node, such as starting, stopping, or restarting them.

Role of the API server

The API server is the central hub of the Kubernetes control plane that exposes the Kubernetes API and processes all requests from clients and other components. The API server performs the following roles:

- It validates and configures the data for the API objects, such as pods, services, deployments, etc.
- It updates the etcd store with the latest state of the API objects.
- It provides a RESTful interface for clients to access and manipulate the API objects.
- It handles the authentication and authorization of clients and enforces admission control policies on requests.
- It supports various features such as versioning, discovery, aggregation, webhooks, etc.

The API server is designed to be scalable and extensible. You can run multiple instances of the API server and balance the traffic between them. You can also extend the API server by adding custom resources and controllers.

Basic Commands

1. **View cluster information:** kubectl cluster-info
2. **Check version:** kubectl version
3. **View API resources:** kubectl api-resources
4. **View component status:** kubectl get componentstatuses

Resource Management

1. **List resources:**
    1. kubectl get pods
    2. kubectl get services
    3. kubectl get deployments
2. **View resource details:**

    kubectl describe pod <pod-name>
3. **Apply a configuration:**

    kubectl apply -f <filename>.yaml
4. **Delete resources:**

    kubectl delete pod <pod-name>

    kubectl delete -f <filename>.yaml

Pods

1. **Run a pod:**

    kubectl run <pod-name> --image=<image-name>
2. **View pod logs:**

    kubectl logs <pod-name>
3. **Execute commands inside a pod:**

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

Deployments

1. **Create a deployment:**

   ```
   kubectl create deployment <deployment-name> --image=<image-name>
   ```

2. **Scale a deployment:**

   ```
   kubectl scale deployment <deployment-name> --replicas=<number>
   ```

3. **Update a deployment:**

   ```
   kubectl set image deployment/<deployment-name> <container-name>=<new-image>
   ```

4. **View deployment rollout status:**

   ```
   kubectl rollout status deployment/<deployment-name>
   ```

5. **Rollback a deployment:**

   ```
   kubectl rollout undo deployment/<deployment-name>
   ```

Services

1. **Expose a deployment as a service:**

   ```
   kubectl expose deployment <deployment-name> --type=<service-type> --port=<port>
   ```

2. **List services:**

   ```
   kubectl get services
   ```

Namespaces

1. **List namespaces:**

   ```
   kubectl get namespaces
   ```

2. **Switch namespace:**

   ```
   kubectl config set-context --current --namespace=<namespace>
   ```

3. **Create a namespace:**

   ```
   kubectl create namespace <namespace-name>
   ```

Configuration and Debugging

1. **View current configuration:**

   ```
   kubectl config view
   ```

2. **View logs of a node or pod:**

```
kubectl logs <pod-name>
```

3. **Debug a pod interactively:**

```
kubectl debug <pod-name> --image=<debug-image>
```

4. **Explain a resource:**

```
kubectl explain <resource-type>
```

Monitoring and Metrics

1. **Top nodes or pods (requires Metrics Server):**

```
kubectl top nodes
kubectl top pods
```

2. **List events:**

```
kubectl get events
```

YAML Management

1. **Generate YAML for a resource:**

```
kubectl get pod <pod-name> -o yaml
```

2. **Edit a resource:**

```
kubectl edit <resource-type>/<resource-name>
```

3. **Replace a resource:**

```
kubectl replace -f <filename>.yaml
```