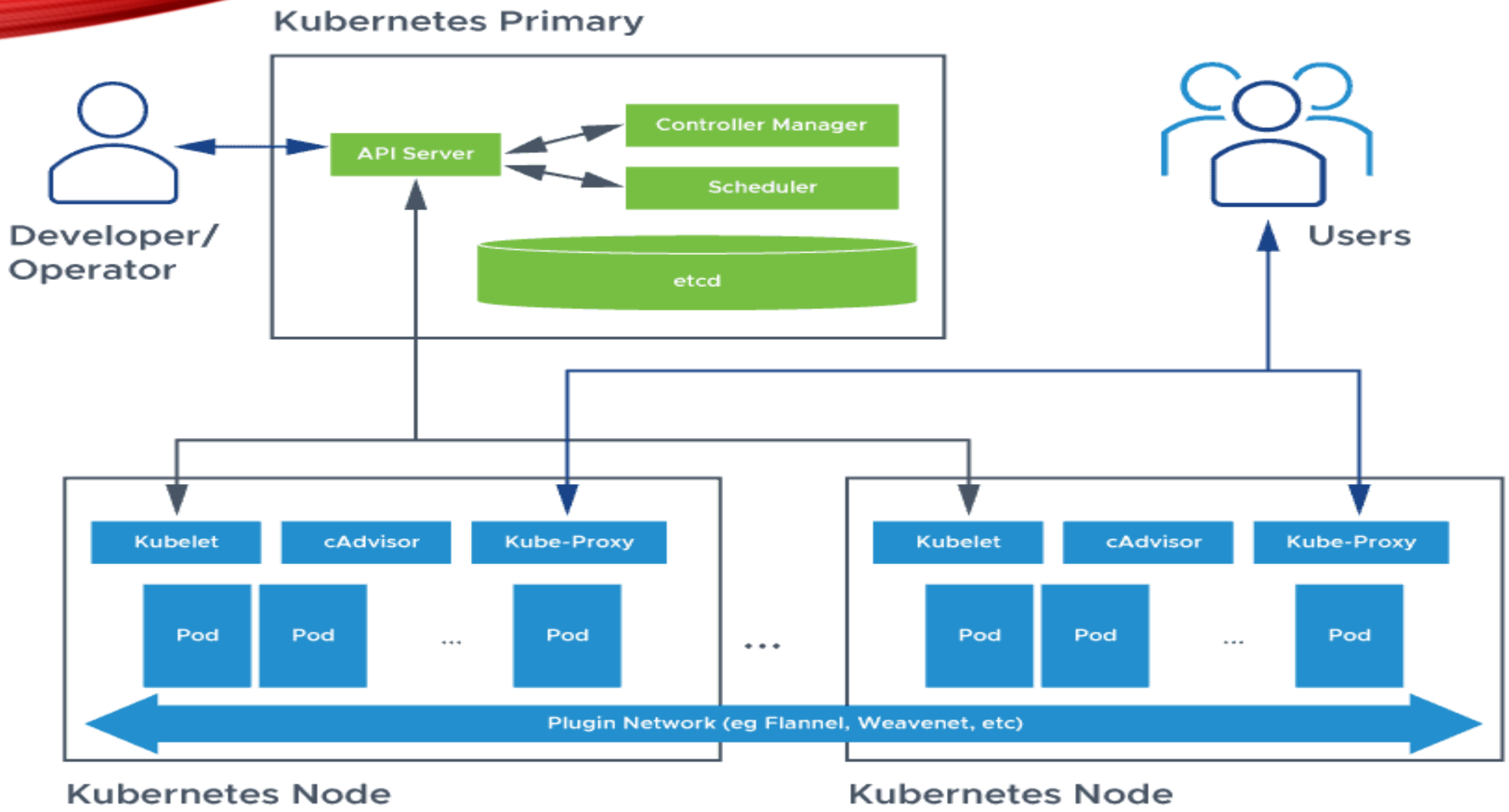
INTRODUCTION TO ORCHESTRATION

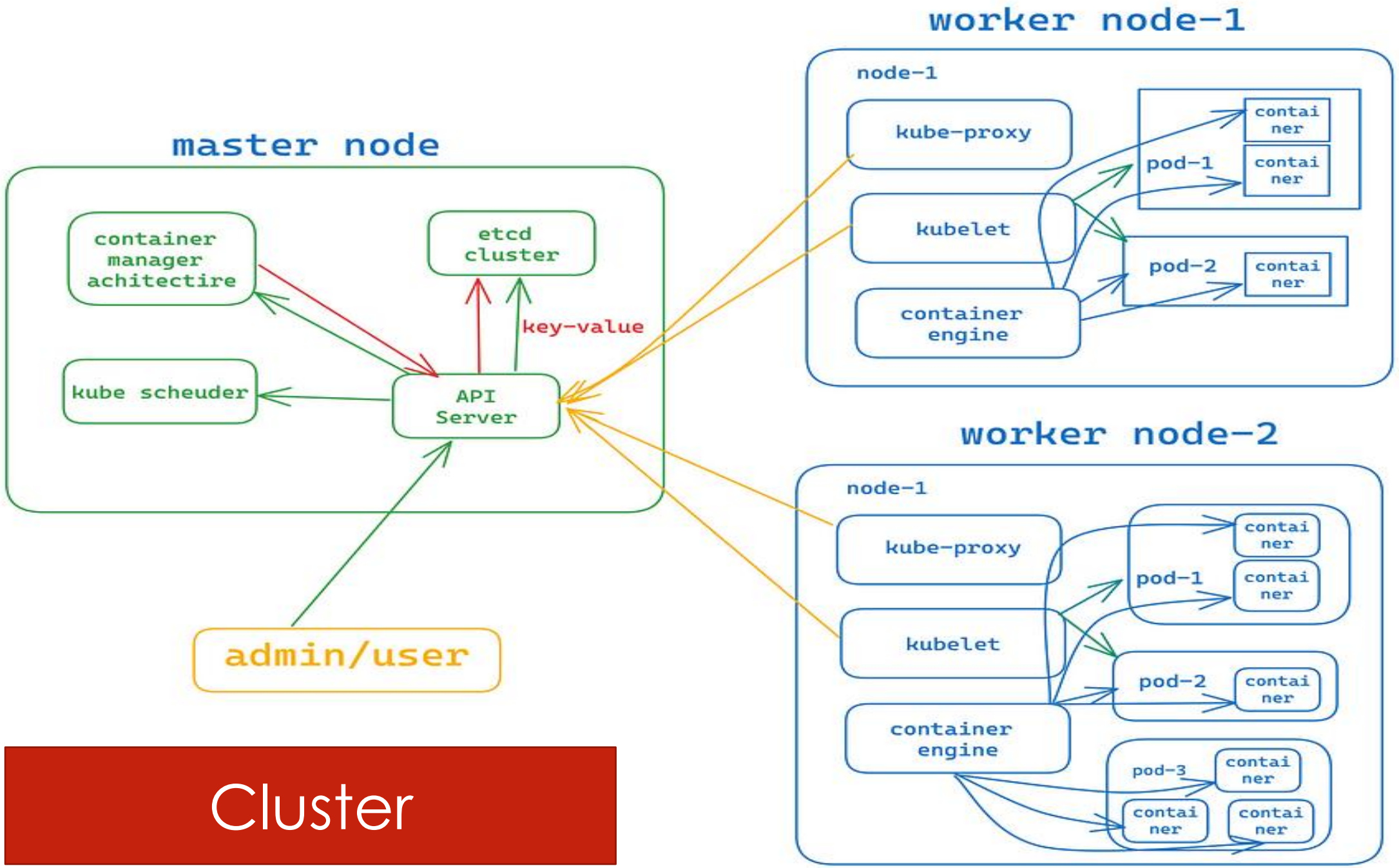
Kubernetes

KUBERNETES

- Kubernetes, an open-source container orchestration platform, is designed to automate the deployment, scaling, and operation of application containers.

KUBERNETES ARCHITECTURE




worker node-1
node-1
kube-proxy
kubelet
container engine
pod-1
pod-2
contai ner
contai ner
contai ner
contai ner
master node
container manager achitectire
etcd cluster
key-value
kube scheuder
API Server
admin/user
worker node-2
node-1
kube-proxy
kubelet
container engine
pod-1
pod-2
pod-3
contai ner
contai ner
contai ner
contai ner
contai ner
contai ner

Cluster

KEY COMPONENTS

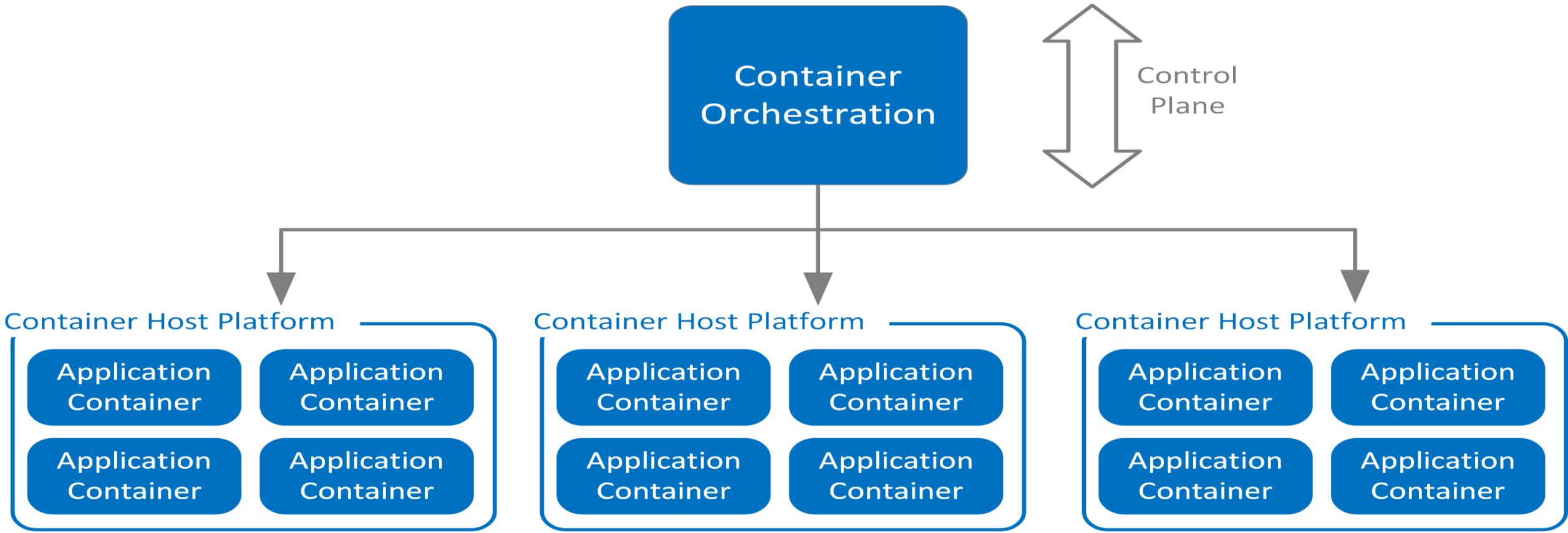
Master Node: Controls the cluster, responsible for the management of the Kubernetes cluster.

- **API Server**: The front-end for the Kubernetes control plane. It exposes the Kubernetes API, acting as the main management point for the cluster.
- **Scheduler**: Assigns workloads to specific nodes based on resource availability and policies.
- **Controller Manager**: Runs controller processes to regulate the state of the cluster, ensuring that the desired state matches the current state.
- **etcd**: A consistent and highly-available key-value store used as Kubernetes' backing store for all cluster data.

Worker Nodes: Run containerized applications. Each node has the necessary services to run pods and is managed by the master node.

- **Kubelet**: An agent that ensures containers are running in a pod. It communicates with the master node and manages the containers' lifecycle.
- **Kube-proxy**: Maintains network rules on nodes, enabling communication to and from pods within the cluster.
- **Container Runtime**: manages and runs components required to run containers
- Pod: Represents a single instance of an application

CONTAINER ORCHESTRATION



KUBERNETES API

- The Kubernetes API is a powerful interface that allows users to interact with and manage their Kubernetes clusters.
- It provides a RESTful interface for querying and manipulating the state of various Kubernetes objects, such as pods, services, deployments, and more.

KEY FEATURES OF KUBERNETES API

- **Resource-Oriented Design**:
  - The Kubernetes API is structured around resources, such as Pods, Services, Nodes, ConfigMaps, and more.
  - Each resource has a corresponding URL endpoint, allowing for operations like create, read, update, and delete (CRUD).
- **RESTful Endpoints**:
  - The API uses standard HTTP verbs: GET (retrieve), POST (create), PUT (update), PATCH (partially update), DELETE (remove).
  - URLs are structured in a hierarchical format, e.g., /api/v1/namespaces/{namespace}/pods.

KEY FEATURES OF KUBERNETES API

- **API Versions**:
  - The Kubernetes API is versioned to maintain compatibility. Common versions include v1, v1beta1, and v1alpha1.
  - Different API groups (e.g., core, apps, batch) may have different versions.

COMMON API RESOURCES

- **Pods**:
  - Smallest and simplest Kubernetes object, representing a single instance of a running process.
- **Services**:
  - Abstract way to expose an application running on a set of Pods as a network service.
- **Deployments**:
  - Provide declarative updates to applications, managing ReplicaSets to ensure the desired number of Pods are running.

COMMON API RESOURCES

- **ConfigMaps**:
  - Store configuration data in key-value pairs.
- **Secrets**:
  - Store sensitive data, such as passwords, OAuth tokens, and SSH keys.
- **Nodes**:
  - Represent a worker node in the cluster.

INTERACTING WITH THE API

- **kubectl**:
  - The primary command-line tool for interacting with the Kubernetes API.
  - Commands like kubectl get, kubectl create, kubectl delete, etc., interact with the API to manage resources.

INTERACTING WITH THE API

- **Client Libraries**:
  - Kubernetes provides client libraries for different programming languages (e.g., Go, Python, Java, JavaScript).
  - These libraries wrap the RESTful API calls, making it easier to programmatically interact with Kubernetes.
- **Direct HTTP Calls**:
  - Users can make direct HTTP requests to the API server endpoints.
  - Authentication is required, typically using tokens, certificates, or other supported methods.

CREATING AND MANAGING CLUSTER

- Create clustuer using Minikube
- Download minikube installer
- Install
- Once instalation done.

    minikube version

    minikube start

Thi command will start kubernetes cluster.

CHECK BASIC COMMANDS

- On the master node, enter the following command to review cluster information:
  - kubectl cluster-info
- On the master node, enter the following command to view complete cluster information
  - kubectl cluster-info dump
- Use the following command to create a namespace:
  - kubectl create namespace firstnamespace

- Confirm the creation of the new namespace with the following command:
  - kubectl get namespaces
- To view the cluster configuration, use the command below:
  - kubectl config view
- Run the following command to view the current cluster:
  - kubectl config current-context
- To identify the API server, execute and copy the 127.0.0.1:8080 port as shown below:
  - kubectl proxy --port=8080

CONFIGURING PODS IN CLUSTER

- Create YAML file: vi pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
   name: my-pod
spec:
   containers:
   - name: my-container
     image: nginx:latest
     ports:
     - containerPort: 80
```

- Apply the updated Pod Definition YAML file:
  - kubectl apply -f my-pod.yaml
- Verify the Pod is running:
  - kubectl get pods

KUBERNATES SERVICE

- To access the Nginx container running in your AKS cluster, you can expose it using a Kubernetes service.
- Create a file named my-service.yaml

```
PS /home/sfj> cat my-service.yaml
apiVersion: v1
kind: Service
metadata:
   name: my-service
spec:
   type: LoadBalancer
   selector:
      app: my-nginx
   ports:
   - protocol: TCP
     port: 80
     targetPort: 80
```

MODIFY THE POD TO MATCH SERVICE

- Update my-pod.yaml to include a label that matches the selector in the service.

```
PS /home/sfj> cat my-pod.yaml
apiVersion: v1
kind: Pod
metadata:
   name: my-pod
   labels:
         app: my-nginx
spec:
   containers:
   - name: my-container
     image: nginx:latest
     ports:
     - containerPort: 80
```

- Apply the updated Pod Definition YAML file
  - kubectl apply -f my-pod.yaml
- Deploy the service to expose the pod:
  - kubectl apply -f my-service.yaml
- Check service status:
  - kubectl get services
- Copy external Ip address and check in browser for running service

```
PS /home/sfj> kubectl get services
NAME          TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)        AGE
kubernetes    ClusterIP      10.0.0.1       <none>           443/TCP        38h
my-service    LoadBalancer   10.0.49.202    172.179.99.34    80:31060/TCP   35s
nginx         LoadBalancer   10.0.188.58    172.179.147.5    80:31882/TCP   38h
```

RUNNING CONTAINERS IN PODS

- A Pod is the smallest and simplest Kubernetes object.
- It represents a single instance of a running process in your cluster.
- Pods are designed to host one or more containers that share the same network namespace, storage, and specifications.
- You run your application containers inside Pods.
- Each Pod has its own IP address, and containers within a Pod can communicate with each other using localhost.
- Pods can be created, destroyed, and recreated by Kubernetes, especially if part of a ReplicaSet, Deployment, or other higher-level abstraction.

RUNNING SERVICES IN PODS

- A Service in Kubernetes is an abstraction that defines a logical set of Pods and a policy by which to access them.
- Services provide stable network endpoints to Pods, allowing you to expose your applications internally or externally.
- Services are used to expose Pods to other Pods within the cluster, or to external users.
- They can load balance traffic across multiple Pods.
- Kubernetes supports several types of Services, including
  - ClusterIP
  - NodePort
  - LoadBalancer

PRACTICAL USE CASE

Pods:

- You deploy an application by creating Pods.
- For example, deploying an Nginx server involves creating a Pod with an Nginx container.

Services:

- You expose the Nginx application to the internet or within the cluster by creating a Service that targets the Nginx Pod.

LET'S SUMMARIZE

- Pods run the actual application containers and manage their lifecycle
- while Services provide stable network endpoints to access those Pods, handle load balancing, and expose the application to other components or external clients.

DEPLOYMENT

- A Deployment in Kubernetes is a higher-level abstraction that manages the lifecycle of Pods and ReplicaSets.

DEPLOYMENT FEATURES

- **Declarative Updates:** You declare the desired state of your application, and the Deployment controller makes the necessary changes to achieve that state.
- **Scaling:** Deployments allow you to scale the number of replicas of your application up or down.
- **Rolling Updates:** You can update your application with zero downtime by rolling out updates incrementally.
- **Rollbacks:** If an update causes issues, you can roll back to a previous stable state.
- **Self-Healing:** If a Pod fails, the Deployment controller replaces it automatically.

CREATING AND CONFIGURING A DEPLOYMENT

- Create a Deployment YAML File:
- Nano deployment.yaml

```
PS /home/sfj> cat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.17.4
        ports:
        - containerPort: 80
```

- Apply the Deployment:
  - kubectl apply -f deployment.yaml
- Verify the Deployment:
  - kubectl get deployments
  - kubectl get pods
- Deployment entire Description
  - kubectl describe deployment nginx-deployment
- Scaling the Deployment:
  - kubectl scale deployment/nginx-deployment --replicas=5
  - You can see the 5 replica running in deployment

UPDATE DEPLOYMENT

```
PS /home/sfj> cat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.18.0 #Updated Image
        ports:
        - containerPort: 80
```

- changing the container image
- modify the YAML file: nano deployment.yaml

- Apply the updated YAML file:
  - kubectl apply -f deployment.yaml
- Rolling Back a Deployment:
  - kubectl rollout undo deployment/nginx-deployment
- Monitoring the deployment
  - kubectl rollout status deployment/nginx-deployment
- Check the deployment description which will show the version rollbacked
  - kubectl describe deployment nginx-deployment