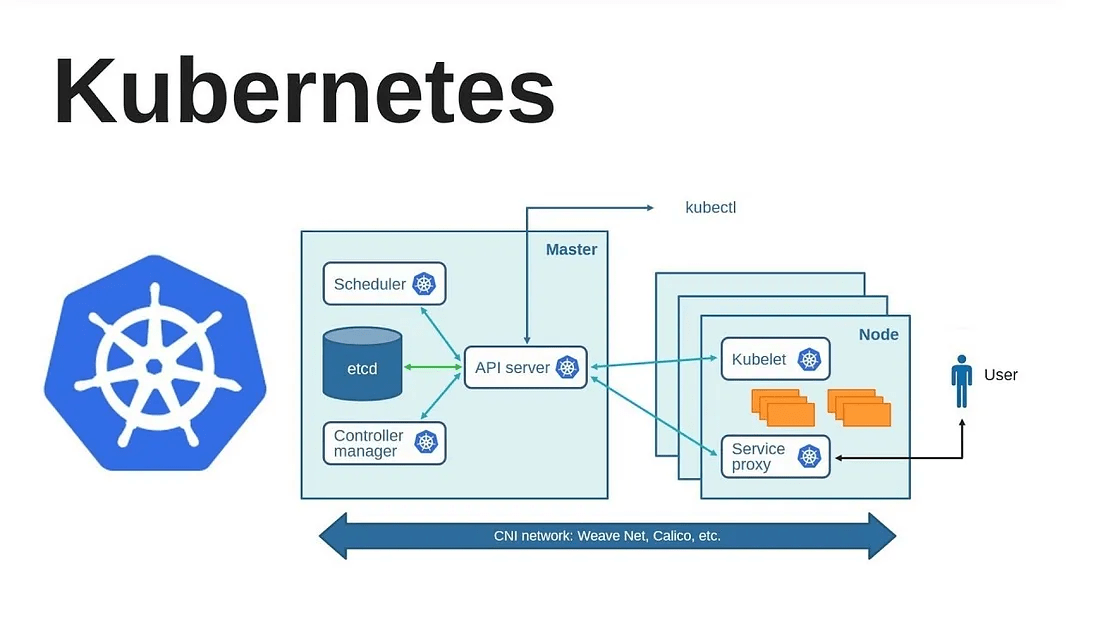
### **Kubernetes Ultimate Notes**

### **Core Concepts**

* Monolithic vs Microservices
* Kubernetes Architecture
* Kubectl
* Pods
* Namespaces
* Labels
* Selectors
* Annotations

|  |  |  |
| --- | --- | --- |
| **Feature** | **Monolithic Architecture** | **Microservices Architecture** |
| **Definition** | Single, unified application | Collection of small, independent services |
| **Codebase** | One large codebase | Multiple smaller, independent codebases |
| **Deployment** | Entire app is deployed at once | Each service can be deployed independently |
| **Scalability** | Scales as a whole | Scales individual services independently |
| **Development Speed** | Slower for large teams (tight coupling) | Faster with independent teams |
| **Technology Stack** | Usually one common tech stack | Each service can use a different tech stack |
| **Data Storage** | Usually one centralized database | Each service can have its own database |
| **Communication** | Internal function calls | API-based (REST, gRPC, message queues) |
| **Failure Impact** | One failure can crash the entire app | One service failure doesn't affect the whole system |
| **Testing** | Complex due to tight coupling | Easier, test services in isolation |
| **Maintenance** | Becomes harder as the app grows | Easier to manage smaller, isolated services |
| **Example Use Case** | Small/medium businesses, MVPs | Large-scale enterprise apps (e.g., Netflix, Amazon) |



### **Master Node (Control Plane)**

The **Master Node** is responsible for managing the cluster and contains several key components:

* **API Server**: Acts as the entry point for all Kubernetes commands (via kubectl). It processes REST requests and updates the cluster state.
* **Scheduler**: Assigns new workloads (Pods) to the appropriate worker nodes based on resource availability.
* **Controller Manager**: Ensures the desired state of the cluster by managing different controllers (e.g., ReplicationController, NodeController).
* **etcd**: A key-value store that holds all cluster configuration data and the current state.

### **2. Worker Nodes**

Worker nodes are responsible for running application workloads. Each node contains:

* **Kubelet**: An agent that ensures containers are running as expected on the node.
* **Service Proxy (Kube-proxy)**: Handles network communication and routes traffic between Pods.
* **Pods**: The smallest deployable unit in Kubernetes, containing one or more containers.

### **3. Communication Flow**

1. Users interact with Kubernetes using **kubectl**, which communicates with the **API Server**.
2. The **API Server** schedules workloads on worker nodes and manages their state.
3. Nodes communicate over a **CNI (Container Network Interface)**, using networking plugins like **Weave Net, Calico, etc.**

## **Namespaces**

**Purpose**: Logical **partitioning of cluster resources** Used to **separate environments** (e.g., dev, staging, prod) or **teams**.

1. Each namespace is **isolated** from the others.
2. Resources like Pods, Services, ConfigMaps, etc. are **scoped to a namespace**.

## **Labels**

**Purpose**: Key-value **metadata** attached to Kubernetes objects  
 Used to **identify, group, and filter** resources.

## **Selectors**

**Purpose**: Mechanism to **select/filter resources** using their labels.

Used by:

1. **Services** to find Pods
2. **ReplicaSets**, **Deployments**, etc. to manage Pods

### **Workloads**

* Deployments
* StatefulSets
* DaemonSets
* ReplicaSets
* Jobs
* CronJobs

## **Deployments**

* **Purpose**: Manage stateless applications.
* Ensures a **specified number of pod replicas** are running at all times.
* Supports **rolling updates** and **rollbacks**.

## **StatefulSets**

* **Purpose**: Manage **stateful** applications (like databases).
* Each pod gets a **stable name**, **persistent storage**, and **orderly startup/shutdown**.

## **ReplicaSets**

* **Purpose**: Ensure a **fixed number of identical pods** are running.
* ReplicaSet is **used by Deployments internally**.

Use directly only for educational/demo purposes. In real apps, use **Deployments**.

## **DaemonSets**

* **Purpose**: Ensures **one Pod per node** (or more).
* Runs critical background services like:  
  + Log collectors (Fluentd)
  + Monitoring agents (Prometheus node-exporter)

Use when:

* You need the pod to exist **on every node**.

## **Jobs**

* **Purpose**: Run **a task once**, to completion.
* Good for batch jobs like:  
  + Database migration
  + Sending emails
  + Image processing

Use when:

* You want to run something **once and exit** after success.

**6. CronJobs**

* **Purpose**: Run **Jobs on a schedule** (like cron).
* Format: "\*/5 \* \* \* \*" = every 5 minutes

Use when:

* You need to automate **recurring tasks** like backups, reports, cleanups.

### **Networking**

* Cluster Networking
* Services
* Ingress
* Network Policies

## **Cluster Networking**

## **What it is:**

Cluster networking enables **communication between Pods, Services, and external users**.

### **Key Characteristics:**

* Every **Pod gets its own IP**.
* Pods can communicate **across nodes** without NAT.
* Communication is typically **flat and open** (all Pods can talk to all other Pods by default).

### **Enabled by:**

* **Container Network Interface (CNI) plugins** like Calico, Flannel, Cilium, Weave.

### **Storage**

* Persistent Volumes (PV)
* Persistent Volume Claims (PVC)
* StorageClasses
* ConfigMaps
* Secrets

### **Persistent Volumes (PV)**

* A **PersistentVolume** is a piece of storage in the cluster that has been **provisioned by an administrator** or **dynamically provisioned** using a StorageClass.
* It is **independent of Pods** and survives Pod restarts/deletion.

### **Persistent Volume Claims (PVC)**

* A **PVC** is a **request for storage** by a user.
* It specifies size, access mode, etc.
* It **binds to a PV** that matches its requirements.

### **StorageClass**

* A **StorageClass** defines **how a PV should be created** (e.g., SSD, slow disk, encrypted).
* Used for **dynamic provisioning** of volumes.

**ConfigMaps**

* Used to **store configuration data** in key-value format.
* Can be mounted as a file or exposed as environment variables in a Pod.

### **Secrets**

* Like ConfigMaps, but **meant for sensitive data** (e.g., passwords, API keys).
* Stored **Base64-encoded**.

### **Scaling and Scheduling**

* HPA (Horizontal Pod Autoscaler)
* VPA (Vertical Pod Autoscaler)
* Node Affinity
* Taints/Tolerations
* Resource Quotas
* Limits
* Probes

### **Cluster Administration**

* RBAC (Role-Based Access Control)
* Cluster Upgrade
* Custom Resource Definitions (CRDs)

### **Monitoring and Logging**

* Metrics Server
* Logging
* Monitoring Tools

### **Advanced Features**

* Operators
* Helm
* Service Mesh
* Kubernetes API

### **Security**

* Pod Security Standards (PSS)
* Image Scanning
* Network Policies
* Secrets Encryption

### **Cloud-Native Kubernetes**

* Managed Services (EKS, AKS, GKE)
* Cluster Autoscaler
* Spot/Preemptible Nodes

## **Cloud-Native Kubernetes Overview**

Cloud-native Kubernetes leverages **cloud provider features** to enhance automation, scalability, and cost efficiency. This includes **managed services**, **autoscalers**, and **cost-optimized node types**.

|  |  |  |
| --- | --- | --- |
| **Service** | **Cloud** | **Description** |
| **EKS** | AWS | Amazon Elastic Kubernetes Service. AWS manages the control plane; you manage worker nodes (or use Fargate). |
| **AKS** | Azure | Azure Kubernetes Service. Microsoft manages the control plane and can also handle nodes with VM scale sets. |
| **GKE** | Google Cloud | Google Kubernetes Engine. GCP offers both standard and autopilot modes (serverless Kubernetes). |

### **Cluster Autoscaler**

**Purpose**: Automatically scales the number of nodes in a cluster **based on pod resource demands**.

#### **How It Works:**

* Watches for **unschedulable pods** (due to insufficient resources).
* Scales **up** by adding nodes to fit those pods.
* Scales **down** if nodes are **underutilized** and pods can fit on fewer nodes.
* Cloud provider integration is required (e.g., auto scaling groups on AWS or VM scale sets on Azure).

### **Spot / Preemptible Nodes**

These are **cost-effective compute options**, ideal for non-critical or batch workloads.

|  |  |  |
| --- | --- | --- |
| **Type** | **Cloud** | **Key Points** |
| **Spot Instances** | AWS | Up to 90% cheaper, can be interrupted with 2 mins notice. |
| **Preemptible VMs** | GCP | Last up to 24 hrs, cheaper, but can be evicted anytime. |
| **Spot VMs** | Azure | Similar concept; interruptions are possible anytime. |

#### 

* Deploy spot nodes in a **dedicated node group** or taint them.
* Use **Node Affinity** or **Tolerations** to run specific workloads on them.
* Combine with **Cluster Autoscaler** for cost-saving burst compute.

### **Debugging and Troubleshooting**

* kubectl Debugging
* Logs
* Resource Usage Analysis