1 Core Principles of Kubernetes

- **Declarative Management**: You declare the desired state of your application using configuration files, and Kubernetes works to make the actual state match it.
- **Self-Healing**: Kubernetes automatically restarts containers that fail, replaces and reschedules Pods when nodes die, and handles failures gracefully.
- **Horizontal Scaling**: You can scale your applications up or down with a simple command or automatically based on CPU usage.
- **Automated Rollouts and Rollbacks**: Kubernetes allows you to automate deployments, ensuring zero downtime for updates and providing the ability to instantly roll back if something goes wrong.
- **Workload Isolation**: Namespaces provide a mechanism for isolating groups of resources within a single cluster.

2 Kubernetes Architecture

2.1 The Control Plane (Master Node)

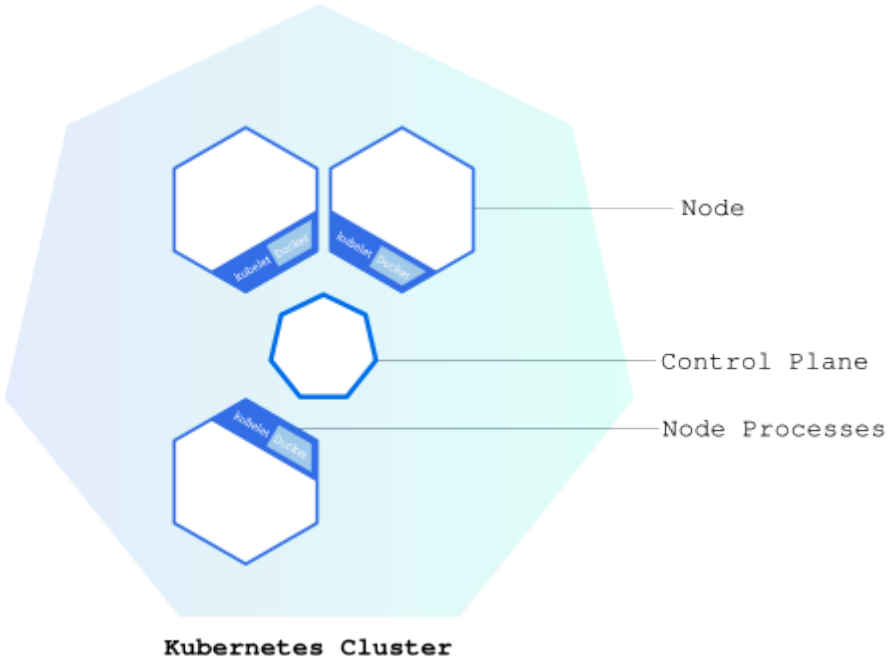
- **API Server**: The central management entity that exposes the Kubernetes API. It is the entrypoint for all cluster communication.
- **etcd**: A consistent and highly-available key-value store that holds all cluster configuration and state data.
- **Scheduler**: Assigns newly created Pods to suitable worker nodes based on resource availability and other constraints.
- **Controller Manager**: Runs various controllers in the background to ensure that the actual state of the cluster matches the desired state.

2.2 The Worker Node

- **kubelet**: The primary agent on each node that ensures containers described in Pods are running and healthy.
- **Container Runtime**: The software responsible for running containers (e.g., Docker, containerd).
- **kube-proxy**: A network proxy on each node that manages networking and load balancing for Services.
- **Applications**: The actual workloads running inside containers, which are organized into Pods.

2.3 Virtual Network

Kubernetes creates a network where every Pod gets its own unique IP address, allowing all Pods in a cluster to communicate directly.



3 Kubernetes Workloads and Objects

3.1 Pods

The Pod is the smallest and simplest deployable unit in Kubernetes. It encapsulates one or more containers, along with shared storage and network resources. Typically, a single application container runs per Pod.

3.2 Controllers: Managing Pods at Scale

- **Deployment & ReplicaSet**: The most common controller for managing stateless applications. A Deployment ensures a specified number of replica Pods are running and handles automated updates and rollbacks.
- **StatefulSet**: Manages stateful applications that require stable network identities and persistent storage (e.g., databases).
- **DaemonSet**: Ensures that a copy of a Pod runs on all (or a selected subset of) nodes in the cluster. This is ideal for logging and monitoring agents.
- **Job & CronJob**: Manages tasks that run to completion. A Job runs a task once, while a CronJob runs a Job on a defined schedule.

3.3 Namespaces

Namespaces provide a way to create virtual clusters within a physical cluster. They are used to isolate resources.

4 Networking in Kubernetes

- **Service**: Since Pods are ephemeral, a Service provides a stable, single endpoint (IP address and DNS name) to access a set of Pods. It automatically load-balances traffic across all replica Pods.
- **Ingress**: Manages external HTTP and HTTPS access to Services within the cluster
- **Network Policy**: Acts as a firewall within the cluster, defining rules that control how Pods are allowed to communicate with each other and with external endpoints.

5 Configuration and Storage

- **ConfigMap**: Stores non-sensitive configuration data (e.g., environment variables, config files) in key-value pairs, decoupling it from the application container.
- **Secret**: Stores sensitive data such as passwords, API tokens, or TLS certificates. They are base64-encoded and can be securely mounted into Pods as files or environment variables.
- **Volume**: Provides a storage directory that is accessible to the containers in a Pod. A Volume's lifecycle is tied to the Pod, allowing data to persist across container restarts.
- **PersistentVolume (PV) & PersistentVolumeClaim (PVC)**: A framework for abstracting storage. A PV is a piece of storage provisioned by an admin, while a PVC is a request for that storage by a user. This decouples the need for storage from the details of the underlying storage system.

6 Managing Resources with YAML

6.1 Basic Structure of a YAML File

Every Kubernetes object definition in YAML has four required top-level fields:

- **apiVersion**: Defines the version of the Kubernetes API to use for creating the object (e.g., `apps/v1`).
- **kind**: Specifies the type of object to be created (e.g., `Deployment`, `Service`, `Pod`).
- **metadata**: Contains data that helps uniquely identify the object, such as its `name` and `labels`.
- **spec**: Describes the desired state for the object, such as the container image, number of replicas, ports, and volumes.

6.2 Example: Deployment YAML

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

6.3 Applying Configurations

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

7 Tools for Kubernetes Development

7.1 Minikube: Local Development

Minikube is a tool that allows you to run a single-node Kubernetes cluster on your local machine. It is ideal for learning, testing, and day-to-day development.

- **Start cluster**: `minikube start`
- **Check status**: `minikube status`
- **Access Dashboard**: `minikube dashboard`
- **Stop cluster**: `minikube stop`

7.2 Helm: The Kubernetes Package Manager

Helm simplifies the deployment and management of applications on Kubernetes. It uses packages called **Charts**, which are collections of pre-configured Kubernetes resources that can be deployed as a single unit.