Kubernetes Concepts and Architecture

1. Introduction

Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. It provides the tools to manage distributed systems across different environments, offering high availability, scalability, and flexibility.

Why Kubernetes?

- **Container Orchestration**: Manages clusters of containers.
- **Automated Deployments**: Facilitates CI/CD pipelines for applications.
- **Scaling and Load Balancing**: Automatically scales the number of containers based on load.
- **Self-Healing**: Restarts failed containers, replaces, kills non-responsive ones, and redistributes containers when nodes die.

2. Kubernetes Architecture

High-Level Overview

Kubernetes consists of a **Control Plane** (Master) and a set of **Nodes** (Workers), each performing specific roles within the cluster.

- **Control Plane**: Manages the cluster.
- **Nodes**: Run containerized applications as pods.

Core Components:

1. **Control Plane (Master Components)**

The control plane is responsible for the management of the Kubernetes cluster. It runs on a master node and includes the following components:

- **API Server (`kube-apiserver`)**:
  - Acts as the front-end for the Kubernetes control plane.
  - Exposes Kubernetes API and processes RESTful requests (CRUD operations on Kubernetes objects like pods, services, deployments).
  - Communicates with other components like etcd, scheduler, and controller manager.
- **etcd**:
  - A distributed key-value store used by Kubernetes to store all cluster data.
  - Stores state information about the entire cluster, including node states, pod specs, and service configurations.
  - Acts as the "source of truth" for the system.
- **Controller Manager (`kube-controller-manager`)**:
  - Ensures the desired state of the system by managing controllers such as node, replication, and endpoint controllers.
  - Monitors the state of the system and makes necessary adjustments to bring the system back to the desired state.
- **Scheduler (`kube-scheduler`)**:
  - Responsible for assigning pods to worker nodes based on available resources and other scheduling constraints (such as affinity, anti-affinity, and taints).
  - It checks for the resource requirements of each pod (CPU, memory) and the availability of those resources on the nodes.

2. **Node (Worker Components)**

Worker nodes are where the actual application workloads (containers) are run. Each node includes:

- **Kubelet**:
  - Agent running on each worker node.

    - Ensures that containers in the form of pods are running in the node as expected.
    - Communicates with the API server to receive and carry out work instructions (e.g., run containers, delete pods).
- **Kube-proxy**:
    - Network proxy that runs on each node.
    - Manages network communication between different pods within the cluster and external clients.
    - Responsible for implementing load balancing and forwarding traffic to the correct pod.
- **Container Runtime**:
    - Software responsible for running the containers on the node (e.g., Docker, containerd, CRI-O).
    - It pulls container images, starts, stops, and manages containers.

3. Key Kubernetes Concepts

1. **Pod**

- The smallest deployable unit in Kubernetes.
- Represents a group of one or more tightly coupled containers.
- Containers within a pod share networking and storage and are always scheduled on the same node.
- Each pod has its own IP address.

2. **Node**

- A worker machine in Kubernetes (physical or virtual) where pods are scheduled and run.
- Each node runs the Kubernetes runtime, `kubelet`, and `kube-proxy` for container orchestration and networking.

3. **Namespace**

- Provides a way to partition a single Kubernetes cluster into multiple virtual clusters.
- Used to group and isolate resources like pods, services, and deployments.

4. **Service**

- Abstracts the way a group of pods are accessed. It provides a single point of entry for accessing these pods.
- Common types:
    - **ClusterIP**: Exposes service within the cluster.
    - **NodePort**: Exposes service on each node's IP at a static port.
    - **LoadBalancer**: Exposes service externally using a cloud provider's load balancer.

5. **Deployment**

- A higher-level abstraction that manages the deployment of pods.
- Ensures that a specific number of pod replicas are running at any given time.
- It automatically updates or rolls back the deployment if a new version of the application is released.

6. **ReplicaSet**

- Ensures that a specified number of pod replicas are running at any given time.
- A deployment automatically manages ReplicaSets and allows for easy scaling of pods.

7. **ConfigMap & Secrets**

- **ConfigMap**: Allows for the external configuration of applications by decoupling configuration artifacts from container images.
- **Secret**: Used to store sensitive information such as passwords, OAuth tokens, or SSH keys.

8. **Ingress**

- Manages external access to services within the cluster, typically HTTP and HTTPS routes.
- Can expose multiple services under the same IP and hostnames, using path-based routing.

9. **Persistent Volumes (PV) & Persistent Volume Claims (PVC)**

- **PV**: A storage resource in a cluster that persists beyond the lifecycle of a pod.
- **PVC**: A request for storage by a user that can be bound to a PV.

10. **Horizontal Pod Autoscaler (HPA)**

- Automatically scales the number of pod replicas based on observed metrics such as CPU utilization.

4. Kubernetes Networking

Kubernetes abstracts networking into several layers:

- **Pod-to-Pod Communication**: Pods can communicate with each other within the cluster using their IP addresses.
- **Service-to-Pod Communication**: Services route traffic to pods based on labels.
- **External Communication**: Ingress and services of type `NodePort` or `LoadBalancer` expose pods externally to the internet.

5. Kubernetes Workflow: A Typical Deployment

1. **Define the Application**:

    - Developers write the Kubernetes configuration files (YAML) to define deployments, services, and other resources.
2. **Submit to the API Server**:
    - The Kubernetes configuration is submitted to the `kube-apiserver` using the `kubectl` command-line tool.
3. **Scheduler Assigns Pods to Nodes**:
    - The scheduler identifies the best-suited node to run the pod based on resource availability.
4. **Kubelet Starts the Pod**:
    - The `kubelet` on the selected node receives instructions to start the container from the container runtime.
5. **Service Exposes Pods**:
    - The service routes network traffic to the running pods and balances traffic across them.

6. Kubernetes Deployment Example

Here is an example of a Kubernetes deployment YAML file for a simple Java Spring Boot application:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: spring-boot-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: spring-boot-app
  template:
    metadata:
      labels:
        app: spring-boot-app
    spec:
      containers:
      - name: spring-boot-container
        image: my-spring-boot-app:latest
        ports:
        - containerPort: 8080
```

Key Parts:

- **replicas**: Ensures there are 3 instances (pods) of the Spring Boot app running.
- **selector**: Matches pods that belong to this deployment.
- **template**: Defines the pod's container image and exposed port.