### **1. Kubernetes Basics and Core Concepts**

* **What is Kubernetes?**
  + Overview of Kubernetes
  + Use Cases and Benefits
* **Kubernetes Architecture Recap**
  + Master Node Components
  + Worker Node Components
* **Kubernetes API Server**
  + Function and Role
  + Interaction with Kubectl
* **Kubernetes Objects and Resources**
  + Pods
  + ReplicaSets
  + Deployments
  + Namespaces
  + Services

### **2. Kubernetes CLI (kubectl) Commands**

* **Basic kubectl Commands**
  + kubectl get, kubectl describe, kubectl logs
* **Managing Resources**
  + Create, Update, Delete Resources
* **Namespace Management**
  + Creating and Switching Namespaces

### **3. Configuring and Managing Pods**

* **Pod Lifecycle**
  + Creation, Running, Termination
* **Pod Configurations**
  + YAML Files, Labels, Annotations
* **Pod Networking**
  + Communication Between Pods
  + Network Policies

### **4. Deployments and ReplicaSets**

* **Creating Deployments**
  + Basic Deployment Configuration
* **Scaling Deployments**
  + Manual and Automatic Scaling
* **Managing ReplicaSets**
  + Rolling Updates and Rollbacks

### **5. Services and Networking**

* **Service Types**
  + ClusterIP, NodePort, LoadBalancer, ExternalName
* **Service Discovery and Load Balancing**
  + Internal vs. External Access
* **Network Policies**
  + Defining and Implementing Policies

### **6. Configuration Management**

* **ConfigMaps**
  + Creating and Using ConfigMaps
* **Secrets Management**
  + Creating and Using Secrets
* **Environment Variables and Volumes**

### **7. Storage Solutions in Kubernetes**

* **Persistent Volumes (PVs) and Persistent Volume Claims (PVCs)**
  + Creating and Managing PVs and PVCs
* **Storage Classes**
  + Default vs. Custom Storage Classes

### **8. Advanced Pod Management**

* **Init Containers**
  + Use Cases and Configurations
* **Sidecar Containers**
  + Patterns and Implementations
* **Pod Disruption Budgets**

### **9. Helm and Package Management**

* **Introduction to Helm**
  + Helm Charts, Repositories, and Releases
* **Using Helm to Deploy Applications**
  + Installing, Upgrading, and Rolling Back Helm Charts

### **10. Monitoring and Logging**

* **Monitoring Tools**
  + Prometheus, Grafana Integration
* **Logging Solutions**
  + Fluentd, ELK Stack Integration

### **11. Security and Best Practices**

* **Pod Security Policies**
  + Implementing and Managing Policies
* **Role-Based Access Control (RBAC)**
  + Roles, RoleBindings, and ClusterRoles
* **Security Contexts and Network Policies**

### **12. Continuous Integration and Continuous Deployment (CI/CD)**

* **CI/CD Pipelines in Kubernetes**
  + Tools and Integration
* **Automating Deployments**
  + GitOps with ArgoCD, Flux

### **13. Kubernetes Networking**

* **Network Models and Concepts**
  + Overlay Networks, CNI Plugins
* **Ingress Controllers**
  + Setting Up and Managing Ingress Rules

### **14. Advanced Kubernetes Topics**

* **Kubernetes Federation**
  + Overview and Use Cases
* **Custom Controllers and Operators**
  + Creating and Managing Operators
* **Kubernetes API Extensions**
  + CRDs and Aggregated APIs

### **15. Kubernetes Troubleshooting and Maintenance**

* **Common Issues and Troubleshooting**
  + Logs Analysis, Debugging Pods
* **Cluster Maintenance**
  + Upgrades, Backup and Restore Strategies

### **16. Scaling Kubernetes and Cloud Providers**

* **Horizontal and Vertical Scaling**
  + Auto-scaling Mechanisms
* **Integration with Cloud Providers**
  + AWS, GCP, Azure Specific Integrations

### **17. Kubernetes in Production**

* **Best Practices for Production Deployments**
  + High Availability, Disaster Recovery
* **Cost Management and Optimization**

### **Basic Structure of a Kubernetes YAML Configuration File**

apiVersion: <api-version>

kind: <resource-kind>

metadata:

name: <resource-name>

namespace: <namespace> # Optional

labels: # Optional

<label-key>: <label-value>

spec:

<spec-field>: # Specification fields specific to the resource type

<subfield>: <value>

...

### **1. Pods**

#### **What is a Pod?**

* **Definition**: A Pod is the smallest and simplest Kubernetes object. It represents a single instance of a running process in the cluster.
* **Components**:
  + **Container**: Runs the application or service.
  + **Storage Volumes**: Shared storage accessible by containers in the Pod.
  + **Networking**: A unique IP address for communication between Pods.

#### **Example**

apiVersion: v1

kind: Pod

metadata:

name: my-app-pod

spec:

containers:

- name: my-app-container

image: nginx:latest

ports:

- containerPort: 80

#### **Use Cases**

* **Basic Application Deployment**:
  + **Example**: Running a simple web server or API service.
  + **Use Case**: Deploying a static website using Nginx.
* **Debugging and Development**:
  + **Example**: Running a single container for debugging or development.
  + **Use Case**: Creating a Pod to test a container image before deploying it with more complex configurations.

### **2. ReplicaSets**

#### **What is a ReplicaSet?**

* **Definition**: A ReplicaSet ensures that a specified number of Pod replicas are running at any given time.
* **Purpose**: Maintains a stable set of replica Pods running at any given time.

#### **Example**

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: my-app-replicaset

spec:

selector:

matchLabels:

app: my-app

replicas: 3

template:

metadata:

labels:

app: my-app

spec:

containers:

- name: my-app-container

image: nginx:latest

ports:

- containerPort: 80

#### **Use Cases**

* **Application Availability**:
  + **Example**: Ensuring multiple instances of a web application are always running.
  + **Use Case**: Scaling a web server to handle increased traffic.
* **Self-Healing Mechanism**:
  + **Example**: Automatically replacing failed Pods.
  + **Use Case**: Ensuring that if a Pod crashes, another one is created to replace it.

### **3. Deployments**

#### **What is a Deployment?**

* **Definition**: A Deployment manages ReplicaSets and provides declarative updates to applications.
* **Purpose**: Automates the process of scaling applications, rolling updates, and rollbacks.

#### **Example**

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-app-deployment

spec:

replicas: 3

selector:

matchLabels:

app: my-app

template:

metadata:

labels:

app: my-app

spec:

containers:

- name: my-app-container

image: nginx:latest

ports:

- containerPort: 80

#### **Use Cases**

* **Application Upgrades**:
  + **Example**: Updating a web server’s image from version 1.0 to 2.0.
  + **Use Case**: Rolling out new features or bug fixes without downtime.
* **Rolling Back Changes**:
  + **Example**: Reverting to a previous version of an application.
  + **Use Case**: Rolling back if a new deployment introduces issues.

### **4. Namespaces**

#### **What is a Namespace?**

* **Definition**: A Namespace is a way to divide cluster resources between multiple users or projects.
* **Purpose**: Helps in resource isolation and organization.

#### **Example**

apiVersion: v1

kind: Namespace

metadata:

name: development

#### **Use Cases**

* **Multi-Tenancy**:
  + **Example**: Creating separate environments for development, testing, and production.
  + **Use Case**: Different teams can work on the same cluster without interfering with each other’s resources.
* **Resource Management**:
  + **Example**: Quotas and limits on resources for different teams.
  + **Use Case**: Setting resource quotas for each team or environment to manage usage.

### **5. Services**

#### **What is a Service?**

* **Definition**: A Service is an abstraction that defines a logical set of Pods and a policy by which to access them.
* **Purpose**: Ensures reliable networking and load balancing between Pods.

#### **Example**

apiVersion: v1

kind: Service

metadata:

name: my-app-service

spec:

selector:

app: my-app

ports:

- protocol: TCP

port: 80

targetPort: 80

type: ClusterIP

#### **Use Cases**

* **Internal Communication**:
  + **Example**: Exposing a web application to other Pods within the same cluster.
  + **Use Case**: Services allow Pods to communicate with each other and with external systems.
* **Load Balancing**:
  + **Example**: Distributing traffic among multiple Pods.
  + **Use Case**: Creating a Service to balance the load for a web application among several instances.

#### **Service Type**

* **ClusterIP**: Default type, only accessible within the cluster.
* **NodePort**: Exposes the Service on each Node’s IP at a static port.
* **LoadBalancer**: Provisioned by cloud providers to expose the Service externally.
* **ExternalName**: Maps the Service to an external DNS name.