DATE:18/07/2024

DAY 4

Kubernet:

Kubernetes, often abbreviated as K8s, is a powerful open-source platform for automating the deployment, scaling, and management of containerized applications. Developed originally by Google, Kubernetes is now maintained by the Cloud Native Computing Foundation (CNCF). Below is a comprehensive guide covering the key concepts, architecture, and best practices for Kubernetes.

1. Introduction to Kubernetes

Kubernetes

Kubernetes is a container orchestration platform that automates the deployment, scaling, and operations of application containers across clusters of machines. It abstracts the underlying infrastructure and provides a unified API for managing containerized applications.

Key Features of Kubernetes

- **Automated Deployment and Scaling:** Automatically deploy, manage, and scale applications.
- **Self-Healing:** Automatically replaces failed containers and restarts applications.
- **Load Balancing**: Distributes traffic across containers.
- **Storage Orchestration:** Manages storage resources for applications.
- **Service Discovery:** Provides DNS names for services and load balances across them.
- **Configuration Management:** Manages configuration and secrets for applications.
- **Rolling Updates:** Performs rolling updates to applications with zero downtime.

2. Kubernetes Architecture

Kubernet components

Component	Description
Master Node	Manages the Kubernetes cluster. Contains components like API server, controller manager, and scheduler.

Kubelet	An agent that runs on each worker node, ensuring that containers are running in Pods
Kbe-Proxy	Maintains network rules for Pod communication and load balancing.
API Server	The entry point for all REST commands used to control the cluster.
Controller Manager	Ensures that the desired state of the cluster is maintained.
Scheduler	Assigns Pods to Nodes based on resource availability and constraints.
Etcd	A distributed key-value store used for storing all Kubernetes cluster data.

Kubernetes Object Lifecycle

Lifecycle Stage	**Description**
Creation	Define objects using YAML or JSON manifests and apply them using kubectl apply.
Update	Modify the configuration and apply changes using kubectl apply.
Scaling	Adjust the number of Pods using kubectl scale.
Deletion	Remove objects using kubectl delete.

3. Core Kubernetes Concepts

Pods

A Pod is the smallest and simplest Kubernetes object. A Pod encapsulates one or more containers.

Example YAML for a Pod:

```
yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
```

```
spec:
  containers:
  - name: my-container
    image: nginx:latest
    ports:
    - containerPort: 80
```

Deployments

A higher-level abstraction for managing a set of Pods. It ensures the desired state is maintained.

Example YAML for a Deployment:

```
yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-container
```

```
    image: nginx:latest
    ports:
    - containerPort: 80
```

Services

Provides a stable IP address and DNS name for a set of Pods.

Example YAML for a Service:

```
yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

ConfigMaps and Secrets

ConfigMaps: Manage configuration data for applications.

Secrets: Store sensitive data such as passwords and tokens.

Example YAML for a ConfigMap:

```
yaml
apiVersion: v1
kind: ConfigMap
```

```
metadata:
  name: my-config
data:
  key1: value1
  key2: value2
```

Example YAML for a Secret:

```
yaml
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
data:
  username: dXNlcg==  # Base64 encoded username
  password: cGFzc3dvcmQ=  # Base64 encoded password
```

Persistent Storage

PersistentVolume (PV): A storage resource in the cluster.

PersistentVolumeClaim (PVC): A request for storage.

Example YAML for a PV:

```
yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-pv
spec:
  accessModes:
```

```
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  hostPath:
    path: "/mnt/data"
```

Example YAML for a PVC:

```
yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Ingress

Manages external access to services, typically HTTP/HTTPS.

Example YAML for an Ingress:

```
yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
```

```
  name: my-ingress
spec:
  rules:
  - host: myapp.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: my-service
            port:
              number: 80
```

Helm Charts

A package manager for Kubernetes, similar to apt for Debian-based systems or yum for Red Hat-based systems.

Basic Commands:

- Install a Chart:

  helm install my-release stable/nginx

- Upgrade a Release:

  helm upgrade my-release stable/nginx

- Uninstall a Release:

  helm uninstall my-release

Helm Chart Example:

yaml

```
apiVersion: v2
name: mychart
description: A Helm chart for Kubernetes
version: 0.1.0

dependencies:
  - name: nginx
    version: 1.16.0
    repository: https://charts.bitnami.com/bitnami

templates:
  - name: deployment.yaml
    apiVersion: apps/v1
    kind: Deployment
    spec:
      replicas: 3
      selector:
        matchLabels:
          app: my-app
      template:
        metadata:
          labels
           app: my-app
        spec:
          containers:
```