Container Orchestration

Question:

What is Container Orchestration? Why is it essential in modern application deployment?

Container Orchestration

Container orchestration is the automated management, deployment, scaling, and operation of containerized applications.

Characteristics -

- Management of Containers
- Automated Operations
- Cluster Management

Container Orchestration

Importance in Modern Application Deployment

- Scalability and Flexibility
- High Availability and Reliability
- Resource Optimization
- Networking and Service Discovery

Container Orchestration

Popular Container Orchestration Tools

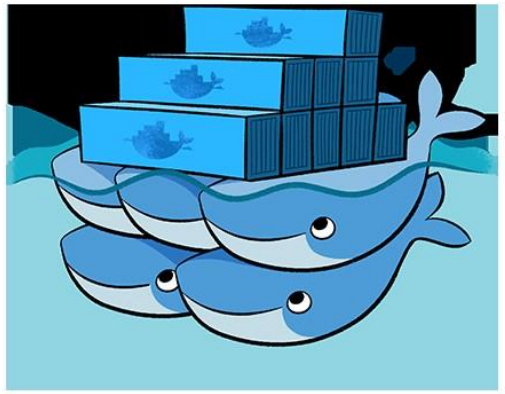
Kubernetes

- Industry Standard
- Open-source
- Robust Features

Container Orchestration

Popular Container Orchestration Tools

Docker Swarm



- Simplicity & Integration
- Small scale deployments

Container Orchestration

Popular Container Orchestration Tools

Amazon ECS(Elastic Container Service)

- AWS Integration
- AWS-Centric Environments

Container Orchestration

Popular Container Orchestration Tools

Apache Mesos

- Flexibility and Scalability
- High level of resource isolation

Container Orchestration

Kubernetes Dominance

- Robust Ecosystem
- Portability and Flexibility
- Extensive Community Support

Container Orchestration

Question:

Explain the key components of Kubernetes architecture?

Container Orchestration

Master Node Components

1. API Server

Primary Management point, validating RESTful API requests from users

Container Orchestration

Master Node Components

2. Scheduler

Assigns newly created pods to nodes

Container Orchestration

Master Node Components

3. Controller Manager

Ensures cluster maintains desired state

Container Orchestration

Master Node Components

4. etcd

Stores cluster state and configuration

Container Orchestration

Worker Node Components

1. Kubelet

Manages pods and their containers

Container Orchestration

Worker Node Components

2. Kube-Proxy

Handles network communication, routes traffic to pods

Container Orchestration

Worker Node Components

3. Container Runtime

Executes containers within pods, storage

Container Orchestration

Interactions and Flow

1. User Interaction

Users interact using `kubectl`

Container Orchestration

Question:

What is Kubernetes namespace and why would you use it?

Container Orchestration

Kubernetes Namespace

Kubernetes namespace is a logical abstraction helps keep objects isolated

Container Orchestration

Kubernetes Namespace

Isolation and Segregation

Resource Scoping

Container Orchestration

Why Kubernetes Namespace

- Multi-Tenancy and Resource Sharing
- Resource Organization
- Access Control and Security
- Testing and Development

Container Orchestration

Kubernetes Namespace

```
kubectl create namespace <name>
```

Container Orchestration

Question:

What is a StatefulSet in Kubernetes and when would you use it?

Container Orchestration

StatefulSet

A StatefulSet in Kubernetes is an API resource used to manage stateful applications, offering features for deploying and maintaining stateful workloads within a Kubernetes cluster

Container Orchestration

Features

Stable and Unique Identifiers

- Assigns stable network identities
- Suitable for MySQL or Kafka

Container Orchestration

Features

Ordered Deployment and Scaling

- Predictable and Ordered rollout
- Sequential deployment of pods in a database cluster

Container Orchestration

Features

Persistent Storage Management

- Manage persistent volumes
- Suitable for Elastic Search, MongoDB

Container Orchestration

Application Suitable for StatefulSet

Databases

- MySQL
- PostgreSQL
- MongoDB
- Cassandra

Container Orchestration

Application Suitable for StatefulSet

Messaging Systems

- Kafka
- RabbitMQ

Container Orchestration

Application Suitable for StatefulSet

Stateful Caches

- Redis
- Memcached

Container Orchestration

Application Suitable for StatefulSet

Distributed Systems

- Elasticsearch
- Zookeeper

Container Orchestration

Example StatefulSet configuration

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mongodb
spec:
  serviceName: "mongodb"
  replicas: 3
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: mongodb-container
        image: mongo:latest
        ports:
        - containerPort: 27017
        volumeMounts:
        - name: mongo-data
          mountPath: /data/db
  volumeClaimTemplates:
  - metadata:
      name: mongo-data
    spec:
      accessModes: ["ReadWriteOnce"]
      resources:
        requests:
          storage: 10Gi
```

Container Orchestration

Question:

What is a Kubernetes service and how does it enable communication between pods?

Container Orchestration

Kubernetes Service

Service is an abstraction that enables communication between various components (usually pods) within a Kubernetes cluster

Container Orchestration

Types of Services

- Cluster IP
- NodePort
- LoadBalancer
- ExternalName

Container Orchestration

Facilitate Communication

- Label-Based Selection
- Stable Virtual IP and Ports
- Internal Load Balancing

Container Orchestration

Yaml file of a Service

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: ClusterIP
```

Container Orchestration

Question:

You're managing a Kubernetes cluster hosting multiple micro services, each handling different functionalities of an e-commerce application. How to enable external access to these services?

Container Orchestration

Ingress

Ingress is an API object used to manage external access to services within a Kubernetes cluster

Container Orchestration

Features of Ingress

- Path-Based Routing
- Hostname-Based Routing
- SSL Termination
- Load Balancing

Container Orchestration

- Ingress Resources
- Ingress Controllers

Container Orchestration

YAML file of Ingress Resource

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
spec:
  rules:
    - http:
        paths:
          - path: /app1
            pathType: Prefix
            backend:
              service:
                name: service-app1
                port:
                  number: 80
          - path: /app2
            pathType: Prefix
            backend:
              service:
                name: service-app2
                port:
                  number: 8080
```

Container Orchestration

Question:

What is the purpose of Service Mesh in Kubernetes?

Container Orchestration

Service Mesh

A Service Mesh in Kubernetes serves as a dedicated infrastructure layer to manage service-to-service communication within a cluster.

Container Orchestration

Purpose

- Service Communication Management
- Traffic Observability and Management
- Security and Policy Enforcement

Container Orchestration

Key Components

- Proxy-Based Infrastructure
- Centralized Control Plane

Container Orchestration

YAML config

```
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
metadata:
  namespace: istio-system
spec:
  profile: default
  components:
    - ingressGateways:
        - name: istio-ingressgateway
          enabled: true
    # ... other configuration options
```

Container Orchestration

Question:

How can you monitor the health of pods and nodes in Kubernetes?

Container Orchestration

Liveness Probe

- Checks if the application within a pod is alive or needs a restart

```
livenessProbe:
  httpGet:
    path: /healthz
    port: 8080
```

HTTP Probe

```
livenessProbe:
  tcpSocket:
    port: 8080
```

TCP Probe

```
livenessProbe:
  exec:
    command:
      - cat
      - /tmp/healthy
```

Exec Probe

Container Orchestration

Readiness Probe

- Checks if the container is ready to serve traffic

```
readinessProbe:
  httpGet:
    path: /ready
    port: 8080
```

HTTP Probe

```
readinessProbe:
  tcpSocket:
    port: 8080
```

TCP Probe

```
readinessProbe:
  exec:
    command:
      - /app/isready.sh
```

Exec Probe

Container Orchestration

Question:

One of your pods is experiencing intermittent issues. Explain the steps you'd take to troubleshoot and diagnose the problem using Kubernetes tools and resources?

Container Orchestration

Systematic Approach

Access Pod and Container logs

```
kubectl logs <pod name> <container name>
```

Container Orchestration

Systematic Approach

Check Pod Events

```
kubectl describe <pod name>
```

Container Orchestration

Systematic Approach

Examine Cluster Events

```
kubectl get events
```

Container Orchestration

Systematic Approach

Evaluate Resource Usage

```
kubectl top pod <pod name>
```

Container Orchestration

Systematic Approach

Review Pod Status

```
kubectl get pods <pod name>
```

Container Orchestration

Systematic Approach

Probe Liveness and Readiness

```
kubectl describe pod <pod name> |grep -A2
"Liveness"
```

Container Orchestration

Systematic Approach

Inspect Node Health

```
kubectl describe node <node name>
```

Container Orchestration

Systematic Approach

Access shell inside pod to further investigate

```
kubectl exec -it <pod name> -- /bin/bash
```

Container Orchestration

Systematic Approach

Check the Network Connectivity from the Pod

```
kubectl exec <pod name> -- curl
http://<service IP>
```

Container Orchestration

Question:

How would you manage persistent storage for a stateful application running in Kubernetes, ensuring data persistence and reliability?

Container Orchestration

Persistent Volume(PV)

Storage abstraction in Kubernetes

Dynamically provisioned using Storage Class

Exist independently of pod

Local storage, Networked Storage, Cloud Storage

Container Orchestration

Persistent Volume Claim(PVC)

Request made by user for a Persistent Volume

PVCs bind themselves with a PV

Container Orchestration

Storage Provisioning Process

PV Provisioning

PVC Creation

Binding

Mounting

Container Orchestration

Managing Persistent Storage for StatefulSets

Define Persistent Volume

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysql-pv-volume
spec:
  capacity:
    storage: 10Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  storageClassName: <storage-class>
  hostPath:
    path: /mnt/data/mysql # Example hostPath, replace with appropriate storage
```

Container Orchestration

Managing Persistent Storage for StatefulSets

Define Persistent Volume Claim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
spec:
  storageClassName: <storage-class>
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

Container Orchestration

Managing Persistent Storage for StatefulSets

Deploy a StatefulSet

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql
spec:
  serviceName: mysql
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      volumes:
        - name: mysql-persistent-storage
          persistentVolumeClaim:
            claimName: mysql-pv-claim
      containers:
        - name: mysql
          image: mysql:latest
          volumeMounts:
            - name: mysql-persistent-storage
              mountPath: /var/lib/mysql
          # Other MySQL configuration settings
```

Container Orchestration

Advantages of PVC in a StatefulSet

Data Persistence

Reliability and High Availability

Disaster Recovery

Container Orchestration

Question:

How would you scale a Kubernetes Deployment based on increasing user demand and what considerations would you take into account?

Container Orchestration

Scenario 1:

An ecommerce website which regularly hosts flash sales needs to dynamically scale the infrastructure to handle increased workload

Container Orchestration

Horizontal Pod Auto-Scaler(HPA)

- Increase the number of pods
- Horizontal scaling efficiently handles increased workload
- YAML file to configure HPA
- Seamless integration with other K8S resources
- Level of control over scaling

Container Orchestration

Scenario 2:

A bigdata processing application is handling hefty batch jobs at different times. It needs more compute power only when the jobs are run and less when quieter

Container Orchestration

Cluster Autoscaler

- Add more worker nodes to the cluster
- Nodes add extra compute power
- Scales the cluster down
- Less demand, Less resources, Less costs
- Integration with Cloud Service Providers

Container Orchestration

Scenario 3:

A machine learning model training job is resource-intensive with multiple such jobs there is a high chance that one of the jobs can consume all available resources

Container Orchestration

Resource Requests and Limits

- Kubernetes Resource request
- Fair share of resources

Container Orchestration

Scenario 4:

A critical database should be highly available and fault tolerant even if few nodes in K8S fails

Container Orchestration

Pod Anti-Affinity

- Avoid placing replicas on the same worker node
- No two replicas are on same worker node
- Replicas tend to run without disruption

Container Orchestration

Question:

In a multi node cluster how would you ensure high availability for your applications? Can you explain the concept of pod affinity and node affinity in this context?

Container Orchestration

Pod Affinity

Node Affinity

Container Orchestration

Pod Affinity

- Influences scheduling of pods based on other pods
- Pod co-location
- Reduce latency
- Improve overall application performance

Container Orchestration

Node Affinity

- Influences scheduling of pods based on other nodes
- Deploy on nodes based on specific labels
- Specialized resources or configurations

Container Orchestration

Question:

Discuss the strategies for managing configuration and sensitive information within the Kubernetes, especially when dealing with secrets

Container Orchestration

Kubernetes Secrets

- Base64 encoded key-value pairs
- Mounted as volume in pods
- Exposed and environment variables

Container Orchestration

Service Account Tokens

- Mounts the service account tokens into pods
- Authentication within the cluster
- `/var/run/secrets/kubernetes.io/serviceaccount/token`

Container Orchestration

HashiCorp Vault

- Easy integration with Kubernetes
- Dynamic Secrets
- Automatic Rotation
- Integrate using K8S Auth method

Container Orchestration

Azure Key Vault

- Azure Key Vault Flex volume
- Suitable for organizations using Azure Cloud services

Container Orchestration

Google Cloud Secret Manager

- Fully managed service on Google Cloud
- Google Cloud Secret Manager CSI Driver
- Suitable for organizations using Google Cloud Services

Container Orchestration

AWS Secrets Manager

- Integrate with K8S using Kube2IAM, External Secrets
- Pods can securely retrieve secrets from AWS Secrets Manager
- Suitable for organizations using Amazon Web Services

Container Orchestration

Bitnami Sealed Secrets

- Open source project by Bitnami
- Encrypted secrets are stored in version control
- Sealed secrets controller decrypts and deploy

Container Orchestration

Square's Keywhiz

- Open source tool by Square
- Integrate with K8S using external secrets
- Suitable for organizations looking open source solutions

Container Orchestration

Question:

In the event of a node failure in the kubernetes cluster, how can you ensure that the affected workloads are rescheduled to healthy nodes efficiently?

Container Orchestration

ReplicaSets and Replication Controllers

- Ensures specified number of pods
- Pod and node failures creates new pods
- Always maintains desired replica count

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: example-replicaset
spec:
  replicas: 3
  selector:
    matchLabels:
      app: example
  template:
    metadata:
      labels:
        app: example
    spec:
      containers:
      - name: example-container
        image: example-image
```

Container Orchestration

Pod Anti-Affinity

- Ensures multiple replicas on nodes
- Prevents single node failure

```
affinity:
  podAntiAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
            - key: app
              operator: In
              values:
              - example
        topologyKey: "kubernetes.io/hostname"
```

Container Orchestration

Node Affinity

- Conditional pod scheduling on node
- Pods based on node characteristics

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: example
          operator: In
          values:
          - true
```

Container Orchestration

Horizontal Pod Autoscaling (HPA)

- Pod replicas based on resources/metrics
- Workload distributions across nodes

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: example-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: example-deployment
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 70
```

Container Orchestration

Pod Disruption Budgets (PDB)

- Limit number of simultaneous disruptions
- Prevents multiple replicas eviction

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: example-pdb
spec:
  maxUnavailable: 1
  selector:
    matchLabels:
      app: example
```

Container Orchestration

Taints and Tolerations

- Taints on nodes repel pods
- Tolerations allow pods to tolerate taints

```
spec:
  containers:
  - name: example-container
    image: example-image
  tolerations:
  - key: "example-node"
    operator: "Exists"
    effect: "NoSchedule"
```

Container Orchestration

Question:

What is a Service Mesh? How would you implement service mesh in your current kubernetes cluster?

Container Orchestration

Service Mesh

Is a dedicated infrastructure layer that facilitates, monitors, and secures communication between micro services within a distributed application

Container Orchestration

Service Mesh Components

- Envoy Proxy
- Pilot
- Citadel
- Galley

Container Orchestration

Service Mesh Features

- Traffic Management
- Observability
- Security
- Fault Injection and Resilience

Container Orchestration

Installation Steps

- Download Istio

```
ISTIO_VERSION=<desired_version>
curl -L https://istio.io/downloadIstio | ISTIO_VERSION=${ISTIO_VERSION} sh -
cd istio-${ISTIO_VERSION}
```

Container Orchestration

Installation Steps

- Install Istio Control Plane

```
istioctl install --set profile=default
```

Container Orchestration

Installation Steps

- Enable Istio sidecar injection

```
kubectl label namespace <your-namespace> istio-injection=enabled
```

Container Orchestration

Installation Steps

- Deploy an application

```
kubectl apply -f your-app-deployment.yaml -n <your-namespace>
```

Container Orchestration

Installation Steps

- Create a Virtual Service

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: your-app
  namespace: <your-namespace>
spec:
  hosts:
    - your-app-service
  http:
  - route:
    - destination:
        host: your-app-service
        subset: v1
      weight: 80
    - destination:
        host: your-app-service
        subset: v2
      weight: 20
```

Container Orchestration

Installation Steps

- Apply Virtual Service Config

```
kubectl apply -f your-app-virtualservice.yaml -n <your-namespace>
```

Container Orchestration

Installation Steps

- Monitor Istio Dashboard

```
istioctl dashboard kiali
```