Kubernetes Orchestration Tools Q&A

In-Class Assignment 2

September 30, 2025

1 Question 1: Key Role of Orchestration Tools

Question: Orchestration tools, such as Kubernetes, play a key role in the server infrastructure for the modern applications.

(a) Explain how these tools help manage and scale application servers.

(b) Describe how orchestration tools facilitate automated deployment, scaling, and management of application servers.

Answer:

1.1 (a) How orchestration tools help manage and scale application servers

- **Resource Management**: Automatically allocate and manage computing resources (CPU, memory, storage), ensuring applications get the resources they need
- **Load Balancing**: Distribute traffic across multiple server instances to improve performance and availability
- **Health Monitoring**: Continuously monitor application status and automatically restart failed instances
- **Horizontal Scaling**: Automatically increase or decrease the number of server instances based on load
- **Service Discovery**: Automatically manage network communication and dependencies between services

1.2 (b) How orchestration tools facilitate automated deployment, scaling, and management

- **Declarative Configuration**: Define desired state through YAML files, and the system automatically maintains that state
- **Rolling Updates**: Update application versions without downtime
- **Auto-scaling**: Automatically adjust the number of instances based on CPU usage, memory, or custom metrics
- **Fault Recovery**: Automatically detect and replace failed nodes or containers

- **Configuration Management**: Centrally manage application configurations and secrets

2 Question 2: Difference between Pod, Deployment, and Service

Question: Explain the difference between a Pod, Deployment, and Service.
Answer:

- **Pod**:
  - The smallest deployable unit in Kubernetes
  - Contains one or more tightly coupled containers
  - Shares network and storage resources
  - Has a short lifecycle and can be deleted and recreated at any time
- **Deployment**:
  - A higher-level controller that manages Pod replicas
  - Ensures a specified number of Pod replicas are always running
  - Supports rolling updates and rollbacks
  - Provides declarative application deployment and management
- **Service**:
  - Provides a stable network access point for Pods
  - Routes traffic to backend Pods through label selectors
  - Provides load balancing functionality
  - Maintains consistent IP and DNS names even when Pods are recreated

3 Question 3: Namespace in Kubernetes

Question: What is a Namespace in Kubernetes? Please list one example.
Answer:

A **Namespace** is a virtual cluster concept in Kubernetes used to create multiple logically isolated environments within the same physical cluster. It provides:

- Resource isolation and organization
- Access control and permission management
- Resource quota limitations
- Unique resource names within the namespace

Example:

```
apiVersion: v1
kind: Namespace
metadata:
  name: development
```

Common default namespaces include:

- `default`: Default namespace
- `kube-system`: System components namespace
- `kube-public`: Public resources namespace

4 Question 4: Role of Kubelet and Node Checking Commands

Question: Explain the role of the Kubelet. How do you check the nodes in a Kubernetes cluster? (kubectl command expected)

Answer:

4.1 Role of Kubelet

- Primary agent running on each node
- Manages Pod lifecycle on the node
- Communicates with the API Server to receive Pod specifications
- Monitors container health and reports to the control plane
- Manages container startup, shutdown, and restart
- Performs health checks and resource monitoring

4.2 kubectl commands to check cluster nodes

```
# View all nodes
kubectl get nodes

# View detailed node information
kubectl get nodes -o wide

# View detailed description of a specific node
kubectl describe node <node-name>

# View node status with labels
kubectl get nodes --show-labels
```

5 Question 5: Difference between ClusterIP, NodePort, and LoadBalancer Services

Question: What is the difference between ClusterIP, NodePort, and LoadBalancer services?

Answer:

- **ClusterIP**:
  - Default service type
  - Only accessible within the cluster
  - Assigns an internal cluster IP address
  - Suitable for internal service communication
- **NodePort**:
  - Opens a specific port on each node (30000-32767)
  - Accessible via any node's IP:NodePort
  - Automatically creates ClusterIP
  - Suitable for development and testing environments
- **LoadBalancer**:
  - Creates an external load balancer (requires cloud provider support)
  - Automatically creates NodePort and ClusterIP
  - Provides an externally accessible IP address
  - Suitable for production external access

6 Question 6: Scaling Deployment to 5 Replicas using kubectl

Question: How do you scale a Deployment to 5 replicas using kubectl?

Answer:

```
# Method 1: Using scale command
kubectl scale deployment <deployment-name> --replicas=5

# Method 2: Using patch command
kubectl patch deployment <deployment-name> -p '{"spec":{"replicas":5}}'

# Verify scaling result
kubectl get deployment <deployment-name>
```

7 Question 7: Updating Deployment Image without Downtime

Question: How would you update the image of a Deployment without downtime?

Answer:

```
# Method 1: Using set image command
kubectl set image deployment/<deployment-name> <container-name>=<new-
   image>

# Method 2: Using patch command
kubectl patch deployment <deployment-name> -p '{"spec":{"template":{"
   spec":{"containers":[{"name":"<container-name>","image":"<new-
   image>"}]}}}}'

# Check rolling update status
kubectl rollout status deployment/<deployment-name>

# Rollback if needed
kubectl rollout undo deployment/<deployment-name>
```

8 Question 8: Exposing Deployment to External Traffic

Question: How do you expose a Deployment to external traffic?

Answer:

```
# Method 1: Create NodePort service
kubectl expose deployment <deployment-name> --type=NodePort --port=80

# Method 2: Create LoadBalancer service
kubectl expose deployment <deployment-name> --type=LoadBalancer --
   port=80

# Method 3: Using kubectl create service
kubectl create service nodeport <service-name> --tcp=80:8080

# Method 4: Through Ingress (need to create Service first)
kubectl expose deployment <deployment-name> --port=80
# Then create Ingress resource
```

9 Question 9: Kubernetes Scheduling Decision Process

Question: How does Kubernetes scheduling decide which node a Pod runs on?

Answer:

The Kubernetes scheduler decides which node a Pod runs on through the following steps:

9.1 Scheduling Process

1. **Filtering Phase**: Exclude nodes that don't meet requirements
   - Resource requirements (CPU, memory)
   - Node selectors and affinity rules
   - Taints and tolerations
   - Port conflict checks
2. **Scoring Phase**: Score eligible nodes
   - Resource utilization balance
   - Affinity preferences
   - Image locality
   - Node load balancing
3. **Selection Phase**: Choose the highest-scoring node

9.2 Influencing Factors

- Node resource availability
- Pod resource requests and limits
- Node selectors (nodeSelector)
- Affinity and anti-affinity rules
- Taints and Tolerations

10 Question 10: Role of Ingress and Difference from Service

Question: What is the role of Ingress and how does it differ from a Service?
Answer:

10.1 Role of Ingress

- Manages external HTTP/HTTPS access to services in the cluster
- Provides load balancing, SSL termination, and name-based virtual hosting
- Supports path-based and domain-based routing
- Centrally manages external access rules

10.2 Difference between Ingress and Service

Feature	**Service**	**Ingress**
Layer	L4 (Transport Layer)	L7 (Application Layer)
Protocol	TCP/UDP	HTTP/HTTPS
Routing	Port-based	Path/domain-based
SSL	Not supported	Supports SSL termination
Load Balancing	Simple round-robin	Advanced load balancing
Cost	Each service needs LoadBalancer	Single entry point

10.3 Example Configuration

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