1. 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.

Orchestration tools like Kubernetes help manage and scale application servers through:

\*\*Management Capabilities:\*\*

Declarative Configuration: Define desired state using YAML/JSON manifests

Self-Healing: Automatically restarts failed containers, replaces pods, and reschedules when nodes fail

Service Discovery: Automatically finds and connects services within the cluster

Configuration Management: Centralized management of environment variables, secrets, and config maps

\*\*Scaling Capabilities:\*\*

Horizontal Pod Autoscaling: Automatically scales the number of pod replicas based on CPU/memory usage or custom metrics

Cluster Autoscaling: Automatically adds or removes nodes from the cluster based on resource demands

Manual Scaling: Quick scaling operations through simple commands

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

\*\*Automated Deployment:\*\*

Rolling Updates: Deploy new versions without downtime by gradually replacing old pods with new ones

Blue-Green Deployments: Run two identical environments and switch traffic between them

Canary Releases: Deploy changes to a small subset of users before full rollout

CI/CD Integration: Seamless integration with Jenkins, GitLab CI, GitHub Actions

\*\*Automated Scaling:\*\*

apiVersion: autoscaling/v2

kind: HorizontalPodAutoscaler

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: my-app

minReplicas: 2

maxReplicas: 10

metrics:

- type: Resource

resource:

name: cpu

target:

type: Utilization

averageUtilization: 70

\*\*Automated Management:\*\*

Health Checks: Liveness and readiness probes ensure only healthy containers serve traffic

Resource Management: CPU and memory limits prevent resource starvation

Auto-remediation: Automatic node repair and pod rescheduling

2. Explain the difference between a Pod, Deployment, and Service.

Pod:

Smallest deployable unit in Kubernetes

Can contain one or multiple containers that share storage and network

Ephemeral and mortal - gets recreated if failed

Directly managed by kubelet

Deployment:

Higher-level abstraction that manages Pods and ReplicaSets

Provides declarative updates for Pods

Enables rolling updates and rollbacks

Maintains desired state and self-heals

Service:

Abstract way to expose an application running on a set of Pods

Provides stable IP address and DNS name

Load balances traffic across Pods

Decouples frontend from backend Pods

Relationship:

Deployment → Manages → ReplicaSet → Manages → Pods

Service → Routes traffic to → Pods

3. What is a Namespace in Kubernetes? Please list one example.

A Namespace is a virtual cluster within a physical Kubernetes cluster that provides isolation and scope for names. It's a way to divide cluster resources between multiple users, teams, or projects.

Example: kubectl get namespaces

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

Kubelet Role:

Primary "node agent" that runs on each node

Responsible for:

Registering the node with the API server

Managing Pod lifecycle

Executing health checks

Reporting node and pod status

Mounting volumes

Downloading secrets

Checking Nodes:

# Check all nodes in the cluster

kubectl get nodes

# Get detailed node information

kubectl describe nodes

# Check node resources

kubectl top nodes

# Get nodes with additional information

kubectl get nodes -o wide

5. What is the difference between ClusterIP, NodePort, and LoadBalancer services?

ClusterIP:

Default service type

Exposes the service on a cluster-internal IP

Only accessible within the cluster

Used for inter-service communication

NodePort:

Exposes the service on each Node's IP at a static port

Accessible from outside the cluster using <NodeIP>:<NodePort>

Automatically creates a ClusterIP service

LoadBalancer:

Exposes the service externally using a cloud provider's load balancer

Automatically creates NodePort and ClusterIP services

Most common way to expose services in cloud environments

6. How do you scale a Deployment to 5 replicas using kubectl?

# Scale deployment to 5 replicas

kubectl scale deployment my-deployment --replicas=5

# Alternatively, using the imperative command

kubectl scale deploy my-deployment --replicas=5

# Verify scaling

kubectl get deployment my-deployment

# Check pod replicas

kubectl get pods -l app=my-app

7. How would you update the image of a Deployment without downtime?

# Method 1: Using set image command (triggers rolling update)

kubectl set image deployment/my-deployment my-container=my-app:v2.0.0

# Method 2: Edit deployment directly

kubectl edit deployment my-deployment

# Method 3: Apply updated YAML

kubectl apply -f deployment-updated.yaml

# Monitor the rolling update

kubectl rollout status deployment/my-deployment

# Rollback if needed

kubectl rollout undo deployment/my-deployment

8. How do you expose a Deployment to external traffic?

# Method 1: Create a LoadBalancer service

kubectl expose deployment my-deployment --type=LoadBalancer --port=80 --target-port=8080 --name=my-service

# Method 2: Create a NodePort service

kubectl expose deployment my-deployment --type=NodePort --port=80 --target-port=8080

# Method 3: Using Ingress (more common for HTTP/HTTPS)

# First create a ClusterIP service

kubectl expose deployment my-deployment --port=80 --target-port=8080

# Then create an Ingress resource

YAML:

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: my-app-ingress

spec:

rules:

- host: my-app.example.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: my-deployment

port:

number: 80

9. How does Kubernetes scheduling decide which node a Pod runs on?

Kubernetes scheduler decides node placement based on:

Scheduling Process:

Filtering: Finds feasible nodes that meet requirements

Scoring: Ranks feasible nodes to choose the best fit

Factors Considered:

Resource Requirements: CPU, memory requests and limits

Node Selectors: nodeSelector constraints

Affinity/Anti-Affinity Rules:

Pod affinity/anti-affinity

Node affinity

Taints and Tolerations: Which pods can tolerate which node taints

Resource Availability: Current node capacity and utilization

Pod Topology Spread Constraints: Spread pods across failure domains

10. What is the role of Ingress and how does it differ from a Service?

Service Role:

Provides internal load balancing

Offers stable network endpoint for pods

Basic L4 (TCP/UDP) load balancing

Three types: ClusterIP, NodePort, LoadBalancer

Ingress Role:

Manages external access to services (typically HTTP/HTTPS)

Provides L7 (application layer) load balancing

Offers name-based virtual hosting

Handles SSL/TLS termination

Path-based routing

Requires an Ingress Controller

Service:

- L4 load balancing

- Internal or basic external access

- Kubernetes native

Ingress:

- L7 load balancing

- Advanced HTTP routing

- Requires Ingress Controller

- More features: SSL, rewrites, authentication