. Role of Orchestration Tools like Kubernetes

(a) Kubernetes and similar orchestration tools simplify management and scaling of applications by:

Automatically distributing and running containers across multiple nodes.

Continuously monitoring container health and restarting containers when they fail.

Efficiently balancing workloads to make optimal use of resources.

Supporting horizontal scaling, adding or removing Pods dynamically based on demand.

(b) They enable automated deployment, scaling, and operational management by:

Using Deployment objects to automate updates to applications.

Supporting auto-scaling driven by CPU, memory, or custom metrics.

Providing self-healing mechanisms, restarting failed Pods without manual intervention.

Allowing rolling updates and rollbacks to ensure applications remain available during upgrades.

2. Difference Between Pod, Deployment, and Service

Pod: The most basic unit in Kubernetes, which can contain one or more containers that share resources and network.

Deployment: A higher-level controller that manages a group of Pods, ensures the desired number of Pods is running, and handles updates.

Service: Offers a stable network endpoint (IP and DNS) to access Pods. It enables load balancing and communication both within and outside the cluster.

3. Kubernetes Namespace

A Namespace is a logical partitioning of cluster resources. It allows multiple teams or projects to share a cluster while keeping their resources isolated.

Example: kube-system is a built-in namespace that contains Kubernetes system components.

4. Kubelet and Node Inspection

The Kubelet runs on every cluster node.

It makes sure that containers described in Pod specs are running correctly.

It reports node and Pod status to the Kubernetes API server.

Command to list nodes:

kubectl get nodes

5. Service Types: ClusterIP, NodePort, LoadBalancer

ClusterIP: Default service type; accessible only inside the cluster.

NodePort: Exposes the service on a static port on each node’s IP, making it reachable externally via <NodeIP>:<NodePort>.

LoadBalancer: Integrates with cloud provider load balancers to provide an external, public IP for the service.

6. Scaling a Deployment

To scale a Deployment to 5 replicas:

kubectl scale deployment myapp-deployment --replicas=5

7. Updating Deployment Image Without Downtime

To update a Deployment’s container image while avoiding downtime:

kubectl set image deployment/myapp-deployment myapp-container=myapp:v2

Kubernetes performs a rolling update, gradually replacing Pods with the new version.

8. Exposing a Deployment to External Traffic

To expose a Deployment externally via a LoadBalancer:

kubectl expose deployment myapp-deployment --type=LoadBalancer --port=80

This creates a Service that routes external traffic to the Pods in the Deployment.

9. How Kubernetes Schedules Pods

The scheduler decides which node a Pod runs on by considering:

Required resources (CPU, memory).

Node taints and tolerations.

Node selectors or affinity rules.

Current workload distribution (preferring less busy nodes).

This ensures efficient resource usage and balanced workloads.

10. Ingress vs Service

Ingress: Handles HTTP/HTTPS traffic from outside the cluster, supports advanced routing rules, TLS termination, and multiple host/domain routing.

Difference:

Ingress operates at Layer 7 (HTTP), can route different paths or domains to different Services, and requires an Ingress Controller.

Service works at Layer 4 (TCP/UDP), exposes a single service endpoint, and is native to Kubernetes.