1.（a）Orchestration tools like Kubernetes automate the deployment, scaling, and management of containerized applications. Kubernetes helps manage application servers by:

Automating Deployment: It allows users to describe the desired state of their application servers (e.g., how many instances of an app should run), and Kubernetes ensures the current state matches this desired state.

Scaling: Kubernetes can scale the number of replicas (instances) of a server automatically based on load, resource utilization, or manually. For example, if the traffic increases, Kubernetes can add more replicas to handle the load.

Self-Healing: Kubernetes constantly monitors the health of nodes and containers. If a container crashes, Kubernetes automatically restarts it, ensuring high availability and fault tolerance.

1. (b)

Automated Deployment: Kubernetes uses Declarative Configurations to manage application deployment. You define the desired state (e.g., number of pods, resources) and Kubernetes continuously works to maintain that state.

Scaling: Kubernetes can scale applications horizontally (by adding/removing pods) using the Horizontal Pod Autoscaler based on metrics like CPU or memory usage.

Management: Kubernetes also provides features like automatic updates (via rolling updates) and resource management (limits, quotas), ensuring that applications run efficiently with minimal manual intervention.

1. Pod: The smallest and simplest Kubernetes object, a Pod is a group of one or more containers that share the same network namespace, storage, and specifications. Pods are usually ephemeral and are often replaced by higher-level abstractions like Deployments.

Deployment: A Deployment manages a set of Pods and ensures that the desired number of Pods are running at all times. It handles deployment strategies, like rolling updates, and is used to scale applications by adjusting the number of Pods.

Service: A Service is an abstraction that defines a set of Pods and a policy for accessing them. Services enable communication between different parts of the application and ensure that the client always accesses the correct Pods, even if the underlying Pods change over time.

1. A **Namespace** is a way to partition resources in a Kubernetes cluster. It allows you to create multiple environments or groups within a single cluster. Namespaces provide isolation for resources like Pods, Services, and Deployments.

**Example**: A typical example would be having separate namespaces for different environments, like dev, test, and prod. You can have resources in the prod namespace that are isolated from the dev namespace.

1. The **Kubelet** is an agent running on each node in a Kubernetes cluster. It ensures that the containers specified in the Pod specs are running and healthy. The Kubelet manages the containers and ensures they are started, stopped, and monitored.

To check the nodes in a Kubernetes cluster, use the following command:

kubectl get nodes

1. **ClusterIP**: The default service type in Kubernetes. It exposes the service on an internal IP address within the cluster, meaning it can only be accessed by other services within the same cluster.

**NodePort**: Exposes the service on a static port on each node in the cluster. This allows external traffic to access the service by hitting any node's IP at the given port.

**LoadBalancer**: Provisions an external load balancer (if supported by the cloud provider) to distribute traffic to the service. It creates a public IP and allows traffic from outside the cluster to reach the service.

1. kubectl scale deployment <deployment-name> --replicas=5
2. kubectl set image deployment/<deployment-name> <container-name>=<new-image>:<new-tag>
3. kubectl expose deployment <deployment-name> --type=LoadBalancer --name=<service-name> --port=<port> --target-port=<target-port>
4. Kubernetes scheduling considers multiple factors when deciding where to place a Pod:

Node Resources: The scheduler checks if the node has enough CPU, memory, and other resources required by the Pod.

Affinity/Anti-affinity: You can define rules about how Pods should be placed based on labels or other criteria (e.g., run this Pod on a node with a specific label).

Taints and Tolerations: If nodes have taints (e.g., not suitable for general Pods), Pods with matching tolerations can be scheduled on those nodes.

NodeSelector: You can specify specific nodes where Pods should run using nodeSelector based on labels.

1. Ingress: An Ingress is a collection of rules that allow HTTP/HTTPS traffic to reach Services in a cluster. It acts as an API gateway and handles routing, SSL termination, and load balancing.

Service: A Service defines a stable endpoint (like ClusterIP, NodePort, or LoadBalancer) for accessing a set of Pods.

Difference:

A Service exposes Pods internally or externally via a fixed IP or port.

An Ingress provides HTTP/HTTPS routing for Services and can manage things like URL-based routing, SSL/TLS termination, and other advanced HTTP-based features.