**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 automate the deployment, scaling, and management of containerized applications. They dynamically allocate resources based on demand, ensure high availability through self-healing mechanisms (e.g., restarting failed containers), and simplify scaling by allowing users to easily adjust the number of application instances (replicas).

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

Automated Deployment: Roll out application updates automatically using declarative configurations (e.g., YAML files).

Scaling: Automatically adjust the number of running instances (pods) based on CPU/memory usage or custom metrics.

Management: Monitor health, load-balance traffic, and replace unhealthy instances without manual intervention.

**2. Difference between a Pod, Deployment, and Service**

Pod: The smallest and simplest Kubernetes object. A Pod represents a single instance of a running process in your cluster and contains one or more containers (e.g., Docker containers). It is the basic unit of deployment but is ephemeral (short-lived).

Deployment: A higher-level abstraction that manages Pods. You declare a desired state for your application (e.g., "run three replicas of this Pod template") in a Deployment, and the Deployment controller works to maintain that state. It enables crucial features like rolling updates and rollbacks.

Service: An abstraction that defines a logical set of Pods and a policy to access them. Since Pods are ephemeral and can be replaced, their IP addresses change. A Service provides a stable IP address and DNS name that acts as a permanent endpoint for a group of Pods, effectively load balancing traffic across them.

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

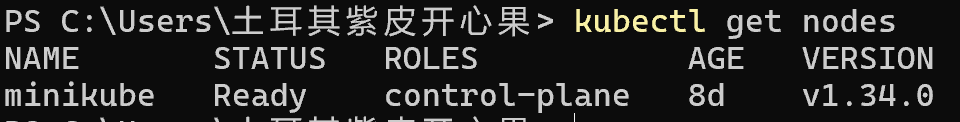
A Namespace in Kubernetes is a virtual cluster backed by the same physical cluster. It provides a mechanism for isolating groups of resources within a single cluster. This is useful for dividing a cluster between multiple users, teams, or projects (e.g., development, staging, production).

Example: `kube-system` - This is the namespace where Kubernetes system components (like the `kube-dns` and `kube-proxy`) are run.

**4. Role of the Kubelet and How to Check Nodes**

Role of the Kubelet: The Kubelet is an agent that runs on each node in the cluster. Its primary role is to ensure that containers are running in a Pod. It takes a set of PodSpecs (pod definitions) provided to it (typically from the API server) and makes sure the containers described in those PodSpecs are running and healthy.

Command to check nodes:kubectl get nodes



**5. Difference between ClusterIP, NodePort, and LoadBalancer Services**

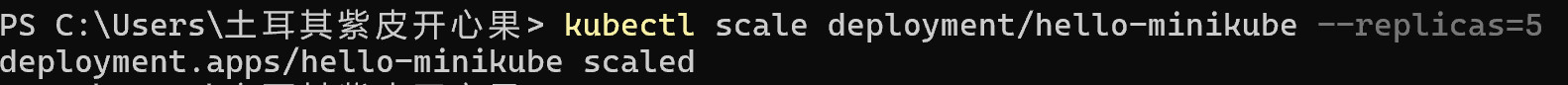
ClusterIP: The default type. The Service is only accessible from within the cluster. It gets a stable, internal IP address and DNS name.

NodePort: Builds on top of ClusterIP. It exposes the Service on each Node's IP at a static port (the `NodePort`, in the 30000-32767 range). The Service is accessible from outside the cluster by contacting `<NodeIP>:<NodePort>`.

LoadBalancer: Builds on top of NodePort. It creates an external load balancer in the cloud provider's infrastructure (e.g., an AWS ELB or a GCP Load Balancer). The load balancer automatically routes external traffic to the `NodePort` and subsequently to the Pods. This is the standard way to expose a service directly to the internet.

**6. How to Scale a Deployment to 5 Replicas**

Use the `kubectl scale` command:**kubectl scale deployment <deployment-name> --replicas=5**



**7. How to Update the Image of a Deployment Without Downtime**

The recommended way is to use a rolling update. You can trigger this by updating the Pod template image in the Deployment. Kubernetes will gradually replace old Pods with new ones, ensuring the application remains available throughout the process.

The command is:

**kubectl set image deployment/<deployment-name> <container-name>=<new-image>:<tag>**

**8. How to Expose a Deployment to External Traffic**

The most common way is to create a Service of type `LoadBalancer` or `NodePort` that targets the Pods managed by your Deployment.

The commmand is :**kubectl expose deployment <deployment-name> --type=NodePort --port=80```**

**9. How Kubernetes Scheduling Decides Which Node a Pod Runs On**

The scheduler assigns pods to nodes based on:

Resource requests/limits (CPU/memory).

Node affinity/anti-affinity rules.

Taints and tolerations.

Pod priority.

**10. Role of Ingress and How It Differs from a Service**

Role of Ingress: An Ingress is an API object that manages external access to the services in a cluster, typically HTTP/HTTPS traffic. It acts as a smart router or entrypoint, providing features like:Host-based routing (e.g., `api.example.com` -> api-service, `app.example.com` -> app-service).

Path-based routing (e.g., `example.com/web` -> frontend-service, `example.com/api` -> backend-service).

TLS/SSL termination.

Difference from a Service:

A Service is a L4 (TCP/UDP) load balancer that provides a stable network endpoint for a set of Pods.An Ingress is a L7 (HTTP/HTTPS) router that sits in front of multiple Services. It is not a type of Service, but it requires an Ingress Controller (a running pod that implements the Ingress rules, like Nginx, Traefik, or an AWS ALB Ingress Controller) to function. A Service of type `LoadBalancer` exposes a single service directly, while an Ingress can expose many services under a single external IP address.