1. （a）

**Automated Container Operations**: Kubernetes automates the deployment, scaling, and management of containers. It groups containers into "pods" and scales them based on demand and resource availability.

**Service Discovery and Load Balancing**: Kubernetes automatically discovers services, performs load balancing, and allocates resources.

**Self-Healing**: Kubernetes monitors health and self-heals by restarting or replicating containers.

Infrastructure Abstraction: Kubernetes handles compute, networking, and storage on behalf of your workloads. This allows developers to focus on their applications without worrying about the underlying environment.

**Optimized Resource Utilization**: By efficiently packaging containers onto nodes based on their requirements, Kubernetes optimizes resource utilization. This helps reduce wasted resources and lowers infrastructure costs.

(b)

**Automated Deployment**: Kubernetes automates application deployment, managing the distribution of containers based on resource availability.

**Autoscaling**: It automatically scales applications based on demand, ensuring optimal resource utilization.

**Self-Healing**: Kubernetes monitors the health of containers and automatically replaces or reschedules them if they fail.

**Load Balancing**: It distributes traffic across containers to ensure no single container is overwhelmed.

**Automated Releases and Rollbacks**: Kubernetes automates the release of new software versions, gradually introducing them into production. If issues arise, it can automatically roll back to a previous stable version, minimizing the impact on users.

2.

Pod: A Pod is the smallest deployable computing unit that can be created and managed in Kubernetes. A Pod can contain one or more containers, which share storage and network resources.

Deployment: A Deployment provides declarative updates for Pods and ReplicaSets. You describe the desired state in a Deployment, and the Deployment controller changes the actual state to the desired state at a controlled rate. You can define a Deployment to create a new ReplicaSet or delete an existing Deployment and use all its resources for a new one.

Service: A Service is an abstract way to expose an application running on a set of Pods as a network service. Kubernetes allows you to use unfamiliar service discovery mechanisms without modifying your applications. Kubernetes provides Pods with their own IP addresses and a DNS name for a group of Pods, and can load balance across them.

3.  
In Kubernetes, namespaces provide a mechanism for isolating groups of resources within a single cluster. Resource names must be unique within a namespace, but not across namespaces.

For example:Suppose you have a development team and a testing team sharing the same Kubernetes cluster. You can create a namespace for each team: development and testing. This way, each team can create and manage resources in their own namespace without interfering with each other's.  
4.The Kubelet is the primary "node agent" that runs on every node. It is responsible for:

Ensuring that the containers described in Pods are running and healthy.

Reporting the status of nodes and Pods to the control plane.

Managing the lifecycle of Pods, such as creating, deleting, and updating Pods.

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

***kubectl get nodes***

This command lists all nodes in the cluster, along with their status, role, age, and version.

5.ClusterIP: This is the default service type. It exposes the service on a cluster-internal IP address. This type of service is only accessible from within the cluster.

NodePort: This service type exposes the service on a static port (NodePort) on each node's IP address. You can contact a NodePort service from outside the cluster using <NodeIP>:<NodePort> .

LoadBalancer: This service type uses a cloud provider's load balancer to expose the service externally. The external load balancer routes traffic to both the NodePort and ClusterIP services.

1. ***kubectl scale deployment my-deployment --replicas=5***

7.To update a Deployment's image without downtime, you can use a rolling update strategy. This is the default update strategy for Deployments in Kubernetes.

When you update a Deployment's image, Kubernetes gradually replaces Pods with the new version. It creates a new Pod at a time, waits for it to be ready, and then deletes the old Pod one at a time. This process continues until all old Pods have been replaced with the new one.

***kubectl set image deployment/my-deployment my-container=my-new-image:latest***

8.***kubectl expose deployment my-deployment --type=NodePort --port=80***

This command creates a Service named my-deployment of type NodePort, which exposes port 80 of the Deployment on a random port on each node.

***kubectl expose deployment my-deployment --type=LoadBalancer --port=80***

This command creates a LoadBalancer Service named my-deployment . If your cluster is running on a cloud provider that supports external load balancers, a load balancer will be provisioned, and you can access your application externally.

9.The Kubernetes scheduler is a control plane process that assigns pods to nodes. The scheduler determines the best node for a pod through a two-step process:

**Filtering:** The scheduler first filters out nodes that don't meet the pod's scheduling requirements. For example, if a pod requests a specific amount of CPU and memory, the scheduler will filter out nodes that don't have sufficient resources.

**Scoring:** After filtering, the scheduler scores the remaining nodes. Scoring is based on a set of predefined priority functions that consider factors such as resource utilization, node affinity, and anti-affinity rules.

The scheduler selects the node with the highest score to run the pod.

10.  
An Ingress is an API object used to manage external access to services in a cluster, typically using HTTP. Ingress can provide load balancing, SSL termination, and name-based virtual hosting.

Below are the **differences** between Ingress and Service:

Services primarily operate at Layer 4 (TCP/UDP) and provide service discovery and load balancing within the cluster.

Ingress operates at Layer 7 (HTTP/HTTPS) and provides more advanced features such as path- and host-based routing, SSL termination, and load balancing.