Kubernetes follows a master-slave architecture where the master node manages the cluster and its components, while the worker nodes (also called minion nodes) run the containerized applications.

The master node consists of several components such as the Kubernetes API server, etcd, scheduler, and controller manager, which work together to manage the cluster state, schedule deployments, and monitor application health.

The worker nodes run the containerized applications and are managed by the Kubernetes control plane. Each node runs a container runtime such as Docker or CRI-O, which enables the creation and management of containers.

The Kubernetes API server acts as the central point of contact for all cluster operations and exposes a RESTful API that can be used to manage the cluster.

etcd is a distributed key-value store that stores the cluster state and configuration data. All nodes in the cluster can access etcd and use it to maintain consistency across the cluster.

The Kubernetes scheduler is responsible for scheduling and deploying containers to the worker nodes based on available resources and constraints.

The Kubernetes controller manager manages different controllers that monitor the state of the cluster and reconcile any differences between the desired and actual state.

Kubernetes also supports add-ons such as DNS, dashboard, and ingress controllers, which provide additional functionality to the cluster.

Container runtime: The container runtime is responsible for creating and managing containers. Kubernetes supports multiple container runtimes, including Docker, CRI-O, and containerd.

Kubelet: Kubelet is the primary agent running on each node that communicates with the control plane to manage the containers running on the node. It ensures that containers are running and healthy and reports their status to the control plane.

Kube-proxy: Kube-proxy is a network proxy that runs on each node and provides a single point of contact for network services to access the containers running on the node. It also handles load balancing across multiple containers.

Pod: A pod is the smallest deployable unit in Kubernetes and represents a group of one or more containers that share the same network namespace and storage volumes. Pods are scheduled and deployed to worker nodes by the control plane.

Container: A container is an isolated runtime environment that contains all the necessary software and libraries to run an application. Containers are managed by the container runtime and can be deployed and managed within pods.

Pods: The smallest deployable unit in Kubernetes that represents a single instance of a running process in the cluster.

ReplicaSets: A ReplicaSet ensures that a specified number of replicas of a Pod are running at any given time.

Deployments: A Deployment is a higher-level abstraction that manages ReplicaSets and provides declarative updates to Pods.

Services: A Service provides a stable IP address and DNS name for a set of Pods, and can load-balance traffic across them.

ConfigMaps: ConfigMaps provide a way to store configuration data that can be consumed by a container at runtime.

Secrets: Secrets provide a way to store sensitive data such as passwords and API keys, and can be mounted as volumes or environment variables in a container.

Namespaces: Namespaces provide a way to divide cluster resources among multiple users or teams.

PersistentVolumes: A PersistentVolume provides persistent storage that can be mounted as a volume in a container.

PersistentVolumeClaims: A PersistentVolumeClaim is a request for storage that can be satisfied by a PersistentVolume.

StatefulSets: A StatefulSet manages the deployment of a set of Pods and provides guarantees about the order in which they are deployed and scaled.

Jobs: A Job creates one or more Pods that perform a specific task, and ensures that the task completes successfully.

CronJobs: A CronJob creates Jobs on a regular schedule, such as every minute or every hour.

ClusterIP: A ClusterIP service provides a stable IP address and DNS name for a set of pods within the cluster, and allows those pods to communicate with each other.

NodePort: A NodePort service exposes a pod to the outside world by opening a specific port on all nodes in the cluster, and forwarding traffic to the corresponding pod.

LoadBalancer: A LoadBalancer service provides load balancing functionality by distributing incoming traffic to a set of pods behind the service.

ExternalName: An ExternalName service maps a service to an external DNS name, allowing pods within the cluster to access resources outside of the cluster.

Ingress: An Ingress service provides a way to route external traffic to multiple services within the cluster based on the requested URL.

NetworkPolicy: A NetworkPolicy provides a way to specify rules that control how pods in the cluster can communicate with each other, based on criteria such as pod labels and IP addresses.

DNS: Kubernetes provides a built-in DNS service that allows pods to discover other pods and services within the cluster using their DNS names.
