Kubernetes

Interview Questions

Q: What is Kubernetes?

A: Kubernetes is an open-source container orchestration platform designed to automate deploying, scaling, and operating application containers. It provides a framework to run distributed systems resiliently, managing the deployment of applications, scaling them as necessary, and managing changes to existing containerized applications.

Q: Explain the architecture of `Kubernetes`.

A: `Kubernetes` architecture consists of a master `node` and worker `nodes`. The master `node` manages the `Kubernetes` cluster and is responsible for the `API server`, `scheduler`, `controller manager`, and `etcd` (the `key-value` store for cluster data). Worker `nodes` run the applications in containers and include `components` like `kubelet` (the agent), `kube-proxy` (networking), and `container runtime` (like `Docker` or containerd).

Q: What are Pods in Kubernetes?

A: A Pod is the smallest deployable unit in Kubernetes, which can contain one or more containers. Pods share the same network namespace, meaning they can communicate with each other using 'localhost,' and they can also share storage volumes. This is useful for tightly coupled applications that need to work together.

Q: What is a `ReplicaSet`?

A: A `ReplicaSet` is a `Kubernetes` resource that ensures a specified `number` of `pod` replicas are running at any given time. It monitors the `state` of the pods and will create or `delete` pods as necessary to maintain the desired `number` of replicas. ReplicaSets are often managed by Deployments, which provide declarative updates to Pods.

Q: What is a `Deployment` in `Kubernetes`?

A: A `Deployment` is a `higher-level` abstraction that manages ReplicaSets and provides declarative updates to applications. It allows you to describe an application's desired `state` and `Kubernetes` will manage the `deployment` process, including `scaling`, `rolling updates`, and rollback capabilities.

Q: How does Kubernetes handle service discovery?

A: Kubernetes provides service discovery through its Service resource. A Service acts as a stable endpoint for accessing a set of Pods, which can change dynamically. Kubernetes assigns a DNS name to the Service, allowing Pods to communicate with it using the service name, and it also handles load balancing between Pods.

Q: What is a ConfigMap in Kubernetes?

A: A ConfigMap is a Kubernetes resource that allows you to decouple configuration artifacts from image content to keep containerized applications portable. ConfigMaps can be used to store non-confidential data in key-value pairs, which can then be consumed by Pods as environment variables, command-line arguments, or configuration files.

Q: What is a PersistentVolume (PV) and PersistentVolumeClaim (PVC)?

A: A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes. A PersistentVolumeClaim (PVC) is a `request` for storage by a user. It specifies size and access modes, and the `Kubernetes control plane` will match the PVC to an available PV.

Q: What is a `Namespace` in `Kubernetes`?

A: A `Namespace` in `Kubernetes` is a way to divide cluster resources between multiple users or teams. It allows for the organization of resources and the implementation of resource quotas, access controls, and more. Namespaces are particularly useful in environments where multiple applications or teams are sharing the same cluster.

Q: Explain the role of the `kubelet`.

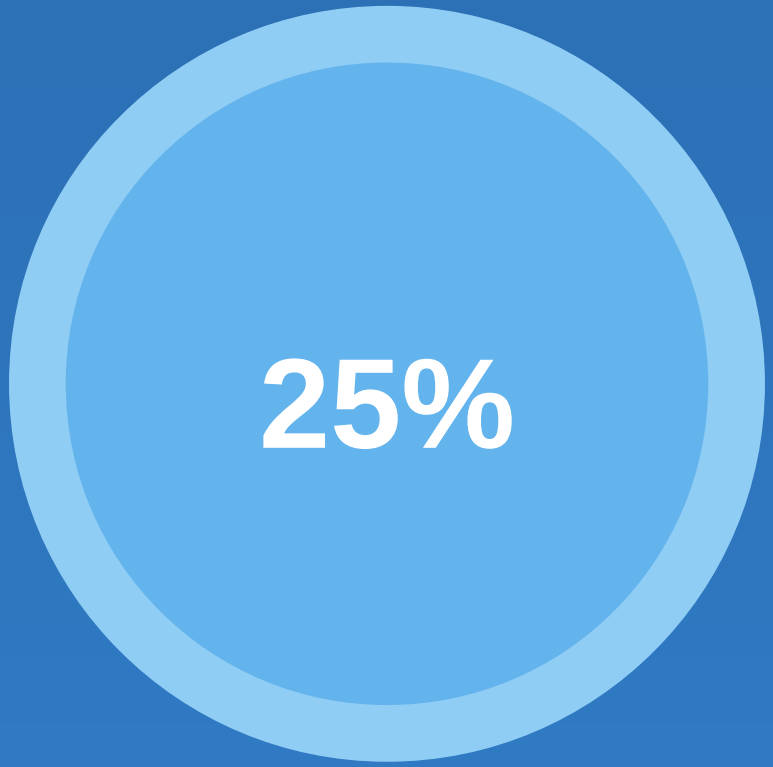
A: The `kubelet` is an agent that runs on each worker `node` in a `Kubernetes` cluster. Its primary responsibility is to ensure that the containers described in the PodSpecs are running and healthy. The `kubelet` communicates with the `Kubernetes API server` to report the status of Pods and to receive updates about desired states.

Q: What are Ingress and Ingress Controllers in Kubernetes?

A: Ingress is a Kubernetes resource that manages external access to services within a cluster, typically HTTP. Ingress Controllers are the components that implement the Ingress resource, providing the rules for routing external traffic to the appropriate service based on the defined Ingress resource.

Q: What is a `StatefulSet` and how does it differ from a `Deployment`?

A: A `StatefulSet` is a `Kubernetes` resource used for managing stateful applications that require persistent storage and unique network identifiers. Unlike Deployments, which are designed for stateless applications, StatefulSets maintain the order and uniqueness of Pods, ensuring stable identities and persistent storage across `Pod` rescheduling.



You're 25% through! Keep going!

Success is built one step at a time.

Q: How can you monitor and log applications in Kubernetes?

A: You can monitor applications in Kubernetes using tools like Prometheus and Grafana for metrics collection and visualization. For logging, you can use a centralized logging solution such as the ELK stack (Elasticsearch, Logstash, Kibana) or Fluentd, which collects logs from all Pods and makes them accessible for searching and analysis.

Q: What is the role of `etcd` in `Kubernetes`?

A: `etcd` is a distributed `key-value` store that acts as the backing store for all cluster data in `Kubernetes`. It stores configuration data, `state` information, and metadata about the cluster. `etcd` is critical for maintaining the desired `state` and configuration of `Kubernetes` resources.

Q: Explain the concept of namespaces in Kubernetes.

A: Namespaces provide a way to divide cluster resources between multiple users or applications. They allow for resource isolation, enabling different teams to work in the same cluster without affecting each other. Namespaces are useful for organizing resources, managing policies, and controlling access.

Q: How can you scale a `deployment` in `Kubernetes`?

A: You can scale a `deployment` in `Kubernetes` by using the `kubectl scale` command or by updating the `deployment`'s specification. For example, you can run `kubectl scale deployment <deployment-name> --replicas=<number>` to change the `number` of `Pod` replicas for that `deployment`. `Kubernetes` will automatically manage the Pods to reach the desired `state`.

Q: What are the differences between a NodePort and a LoadBalancer `service` ?

A: A NodePort `service` exposes a `service` on each `Node`'s IP at a static port, allowing external traffic to access the `service` via any `Node`'s IP and the specified port. A LoadBalancer `service`, on the other hand, provisions an external load balancer (if supported by the cloud provider) that forwards traffic to the `service`, providing a single point of access.

Q: Explain the concept of Helm in Kubernetes.

A: Helm is a package manager for Kubernetes that simplifies the deployment and management of applications on the platform. Helm uses charts, which are pre-configured Kubernetes resources, allowing users to easily install, upgrade, and manage applications in a Kubernetes cluster.

Q: What is the purpose of the `Kubernetes Scheduler`?

A: The `Kubernetes Scheduler` is responsible for assigning Pods to `Nodes` based on resource availability and defined constraints. It evaluates the requirements of each `Pod` and the current `state` of the cluster, making decisions to ensure optimal resource utilization and performance.

Q: How does Kubernetes ensure high availability?

A: Kubernetes ensures high availability through several mechanisms, including replication of Pods, automated health checks, and self-healing capabilities. By using Deployments, StatefulSets, and ReplicaSets, Kubernetes can automatically replace failed Pods and distribute workloads across multiple Nodes.

Q: What are the key components of a Kubernetes architecture?

A: The key components of Kubernetes architecture include the Master Node, which controls the cluster, and the Worker Nodes, which run the application workloads. Key components of the Master Node include the API Server, etcd (the key-value store), the Controller Manager, and the Scheduler. Worker Nodes run the Kubelet, which manages the containers, and the Kube Proxy, which manages network routing.

Q: How does `Kubernetes` handle `load balancing`?

A: `Kubernetes` provides `load balancing` through Services, which abstract access to a set of Pods. By using a `Service`, incoming traffic can be distributed evenly across the Pods it targets, ensuring that no single `Pod` gets overwhelmed. `Kubernetes` also supports external load balancers provided by cloud providers to expose services outside the cluster.

Q: What is a `StatefulSet`?

A: A `StatefulSet` is a `Kubernetes` resource used for managing stateful applications. Unlike a `Deployment`, which manages stateless applications, a `StatefulSet` maintains a sticky identity for each of its Pods, ensuring that they are deployed in a specific order and that their persistent storage is properly retained across rescheduling.

Q: What is a Helm chart?

A: A Helm chart is a package format for `Kubernetes` applications. It contains all the necessary resources and configurations required to deploy an application on `Kubernetes`, including templates for `Kubernetes` manifests, default values, and metadata. Helm simplifies the `deployment` and management of applications by allowing users to easily install, upgrade, or roll back applications.

Halfway there! Every expert was once a beginner.

Q: How does Kubernetes handle storage?

A: Kubernetes abstracts storage using Persistent Volumes (PVs) and Persistent Volume Claims (PVCs). PVs are storage resources in the cluster, while PVCs are requests for those resources by Pods. This abstraction allows for dynamic provisioning and management of storage, enabling applications to use storage resources without being tightly coupled to the underlying infrastructure.

Q: What is Ingress in Kubernetes?

A: Ingress is a Kubernetes resource that manages external access to services within a cluster, typically HTTP/S traffic. It provides routing rules to direct incoming requests to the appropriate services based on the request's host and path. Ingress can also handle SSL termination and provide load balancing.

Q: What is a DaemonSet?

A: A DaemonSet is a `Kubernetes` resource that ensures that a copy of a specific `Pod` runs on all (or a subset of) `nodes` in the cluster. It is commonly used for deploying `system-level` services, such as logging agents or monitoring tools, that need to run on every `node`.

Q: Explain the concept of Pods in Kubernetes.

A: A Pod is the smallest deployable unit in Kubernetes and represents a single instance of a running process in a cluster. A Pod can contain one or more containers that share the same network namespace and storage, allowing for easy communication and data sharing among them.

Q: What is a `ReplicaSet` and how does it differ from a `Deployment`?

A: A `ReplicaSet` is a `Kubernetes` resource that ensures a specified `number` of `pod` replicas are running at any given time. A `Deployment` is a `higher-level` abstraction that manages ReplicaSets and provides additional features such as `rolling updates` and rollback capabilities. Essentially, a `Deployment` creates and manages ReplicaSets.

Q: What is a `Service` in `Kubernetes`, and why is it used?

A: A `Service` in `Kubernetes` is an abstraction that defines a logical set of Pods and a policy to access them. It provides a stable `endpoint` for clients to connect to, enabling `load balancing` and `service discovery`. Services help in exposing applications running in Pods to the network, whether internally or externally.

Q: How does Kubernetes handle scaling, and what are the types of scaling available?

A: Kubernetes supports both manual and automatic scaling. Manual scaling is done using the 'kubectl scale' command to increase or decrease the number of Pod replicas. Automatic scaling can be achieved using the Horizontal Pod Autoscaler (HPA), which adjusts the number of replicas based on observed CPU utilization or other select metrics.

Q: Can you explain the role of `etcd` in `Kubernetes`?

A: `etcd` is a distributed `key-value` store used by `Kubernetes` to store all cluster data, including configuration data, `state` information, and metadata. It acts as the primary data store for the `Kubernetes Control Plane`, enabling high availability and consistency across the cluster.

Q: What is the difference between a `StatefulSet` and a `Deployment`?

A: A `StatefulSet` is a `Kubernetes` resource designed for managing stateful applications that require persistent storage and stable network identities. Unlike Deployments, which are suitable for stateless applications, StatefulSets maintain the order of `deployment` and `scaling`, ensure that Pods are created with unique identifiers, and can provide stable storage using persistent volumes.

Q: Describe how `Kubernetes` manages networking between Pods.

A: `Kubernetes` uses a flat network `model` that allows Pods to communicate with each other using their IP addresses, regardless of the `node` they are running on. Each `Pod` gets its own IP address, and `Kubernetes` provides a network overlay that supports `service discovery`, `load balancing`, and `intra-cluster` communication.

Q: What is Helm, and how does it relate to `Kubernetes`?

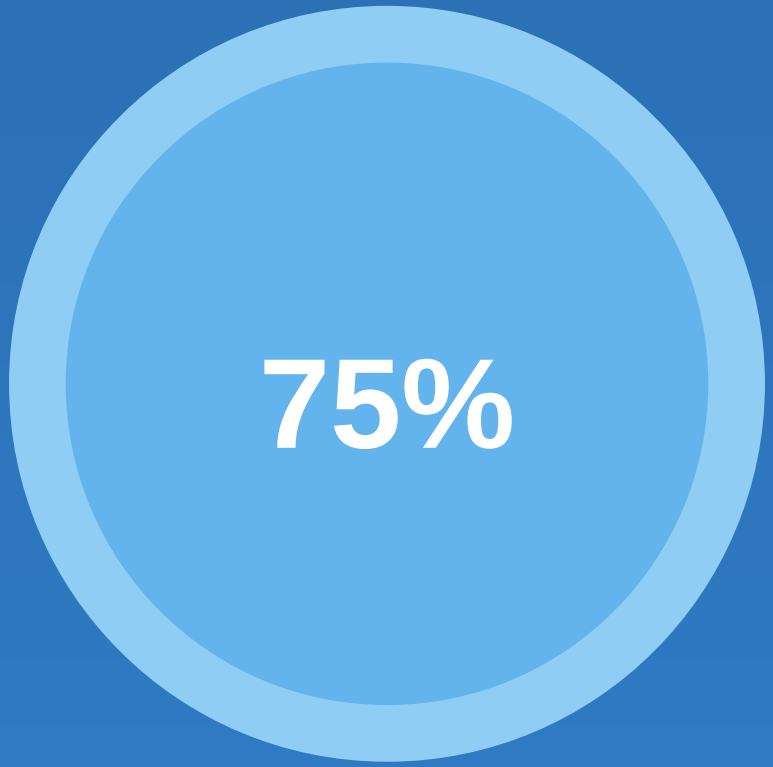
A: Helm is a package manager for `Kubernetes` that simplifies the `deployment` and management of applications by allowing users to define, install, and upgrade complex `Kubernetes` applications using Helm charts. Helm provides templating features to manage `Kubernetes` resources and facilitates version control of deployments.

Q: What is a DaemonSet in Kubernetes?

A: A DaemonSet is a Kubernetes resource that ensures a copy of a specific Pod runs on all (or a subset of) nodes in a cluster. It is typically used for deploying system services or monitoring agents that need to run on every node, such as logging or network monitoring tools.

Q: How do you perform a rolling update in Kubernetes?

A: A rolling update in Kubernetes can be performed using a Deployment. By updating the Deployment's image or configuration, Kubernetes gradually replaces old Pods with new ones without downtime. The update strategy can be configured to control the maximum number of unavailable Pods and the maximum number of new Pods that can be created during the update.



You're at 75%! Almost done, push through and finish strong!

Q: What are `Ingress` resources in `Kubernetes`?

A: `Ingress` resources in `Kubernetes` manage external access to services within a cluster, typically via `HTTP/S`. An `Ingress` `controller` is responsible for processing `Ingress` resources and `routing` traffic based on defined rules. `Ingress` allows for features like SSL termination, `path-based` `routing`, and `host-based` virtual hosting.

Q: What is the purpose of a `Node` in a `Kubernetes` cluster?

A: A `Node` in a `Kubernetes` cluster refers to a worker machine that runs containerized applications as Pods. `Nodes` can be physical or virtual machines and are managed by the `Kubernetes` `Control Plane`. Each `Node` runs a `Kubelet`, which communicates with the `Control Plane`, and a `container runtime` to manage the execution of containers.

Q: What strategies can be used for monitoring and logging in Kubernetes?

A: Common strategies for monitoring and logging in Kubernetes include deploying monitoring solutions like Prometheus for metrics collection and Grafana for visualization, as well as using centralized logging solutions such as ELK Stack (Elasticsearch, Logstash, and Kibana) or Fluentd for log aggregation. These tools can help in observing the performance and health of applications and the cluster.

Q: What are the benefits of using Kubernetes?

A: The benefits of using Kubernetes include automated deployment and scaling, self-healing capabilities (automatic restarts, replication, and scaling), resource optimization, multi-cloud and hybrid cloud flexibility, improved developer productivity through CI/CD integrations, and a strong ecosystem of tools and community support.

Q: What is the purpose of the `kubelet`?

A: The `kubelet` is an agent that runs on each worker `node` in a `Kubernetes` cluster. Its primary `job` is to ensure that the containers described in `pod` specifications are running and healthy. The `kubelet` communicates with the `Kubernetes API server` to `get` the desired `state` of the pods and reports back the current `state`. It manages the lifecycle of containers, including starting, stopping, and monitoring their health.

Q: What are ConfigMaps and Secrets?

A: ConfigMaps and Secrets are `Kubernetes` resources used to manage configuration data. ConfigMaps are used to store `non-sensitive` configuration information in `key-value` pairs, which can be consumed by pods as environment variables or mounted as files. Secrets, on the other hand, are intended to hold sensitive information, such as passwords, tokens, or SSH keys, and are stored in an encoded format to enhance security. They can also be mounted as files or provided as environment variables to containers.

Q: What are DaemonSets?

A: A DaemonSet is a `Kubernetes` resource that ensures that a copy of a specific `Pod` runs on all (or a subset of) `nodes` in the cluster. It is used for deploying `system-level` services, such as log collectors and monitoring agents, that need to run on every `node`. When `new nodes` are added to the cluster, the DaemonSet automatically schedules Pods on those `nodes`.

Q: What is the role of the Kubernetes API server?

A: The Kubernetes API server is a central component of the Kubernetes control plane that exposes the Kubernetes API. It acts as the gateway for all interactions with the cluster, allowing users and components to communicate with the Kubernetes system. The API server processes REST operations and updates the etcd store, ensuring the desired state of the cluster is maintained. It also handles authentication, authorization, and admission control.

Q: How can you perform `rolling updates` in `Kubernetes`?

A: `Rolling updates` in `Kubernetes` can be performed using Deployments, which allow you to update the application without downtime. When you update the `image` or configuration of a `Deployment`, `Kubernetes` gradually replaces old Pods with `new` ones, ensuring that a specified `number` of Pods are always available during the update. You can control the update process using parameters like maxUnavailable and maxSurge to define how many Pods can be taken down or added during the update.

Q: What is a `Service` in `Kubernetes`?

A: A `Service` in `Kubernetes` is an abstraction that defines a logical set of Pods and a policy to access them. It allows for stable networking by providing a persistent IP address and DNS name, enabling `load balancing` and `service discovery`. Services can be of different types, such as ClusterIP (accessible within the cluster), NodePort (accessible externally on a specified port), and LoadBalancer (integrating with cloud providers for external `load balancing`).

Q: What is the role of the `Kubernetes Scheduler`?

A: The `Kubernetes Scheduler` is responsible for assigning Pods to available `nodes` in a cluster based on resource requirements and constraints. It evaluates the Pods' specifications and the current `state` of the `nodes`, considering factors like resource availability, taints, tolerations, and `affinity/anti-affinity` rules to make optimal scheduling decisions.

Q: Explain the concept of Taints and Tolerations.

A: Taints and Tolerations are mechanisms in Kubernetes that control how Pods are scheduled onto nodes. A taint on a node prevents Pods from being scheduled onto that node unless the Pod has a matching toleration. This allows for greater flexibility in managing node resources and ensuring that specific workloads run on designated nodes, helping to manage resource allocation and isolation.

Q: What are the differences between Kubernetes and Docker Swarm?

A: Kubernetes and Docker Swarm are both container orchestration tools, but they have different architectures and features. Kubernetes is more feature-rich and provides a robust ecosystem for managing containerized applications at scale, offering extensive functionality like automated scaling, self-healing, and built-in monitoring. Docker Swarm is simpler and easier to set up, focusing on ease of use and integration with the Docker ecosystem, but lacks many of the advanced features available in Kubernetes.

Thank You!

You've completed all the questions.

GM SUNSHINE