Kubernetes Important interview Questions.

Kubernetes Q&A

1. **What is Kubernetes and why it is important?**
   Kubernetes is an open-source platform used to automate the deployment, scaling, and management of containerized applications. It is like a traffic controller for containerized applications. It ensures that these applications are running efficiently and reliably, by managing their deployment, scaling, and updating processes.
   Kubernetes is important because it makes much easier to deploy and manage complex applications across different environments and infrastructures. By providing a consistent platform for containerized applications, Kubernetes allows developers to focus on building and improving their applications, rather than worrying about the underlying infrastructure. Additionally, Kubernetes helps organizations to achieve greater efficiency, scalability, and flexibility, which can result in significant cost savings and faster time-to-market.

2. **What is difference between docker swarm and Kubernetes?**

Kubernetes	Docker Swarm
1. Kubernetes Setup is complicated but once installed the cluster is very strong.	1. Docker Swarm setup is very simple but the cluster is not very strong.
2. GUI is the Kubernetes Dashboard.	2. There is no GUI.
3. Kubernetes can do auto-scaling.	3. Docker Swarm can't do auto-scaling.
4. Kubernetes can deploy rolling updates, & does automatic rollbacks.	4. Docker Swarm can deploy rolling updates, but not automatic rollbacks.
5. Can share Storage volumes only with other containers in the same pod.	5. Can share Storage volumes with any other containers.
6. It has built-in tools for logging and monitoring.	6. It uses 3rd party tools like ELK stack for logging and monitoring.
7. Manual intervention needed for load balancing traffic between different containers and pods.	7. It does auto load balancing of traffics between containers in the cluster.

3. **How does Kubernetes handle network communication between containers?**
   Kubernetes defines a network model called the container network interface (CNI), but the actual implementation relies on network plugins. The network plugin is responsible for allocating internet protocol (IP) addresses to pods and enabling pods to communicate with each other within the Kubernetes cluster.
   When a pod is created in Kubernetes, the CNI plugin is used to create a virtual network interface for the pod. Each container in the pod is then assigned its own unique IP address within the pod's network namespace. This enables containers within the pod to communicate with each other via localhost, as if they were running on the same host.
   To enable communication between pods, Kubernetes sets up a virtual network overlay using the selected network plugin. Each node in the cluster runs a network agent that communicates with other agents on other nodes to establish the overlay network. This enables communication between containers running in different pods, even if they are running on different nodes.

4. **How does Kubernetes handle scaling of applications?**
   Kubernetes provides built-in mechanisms for scaling applications horizontally and vertically, allowing you to meet changing demands for your application.
   Horizontal scaling, also known as scaling out, involves adding more instances of an application to handle increased traffic. Kubernetes can manage this automatically through the use of ReplicaSet,

which ensure that a specified number of identical pods are running at all times. We can configure the ReplicaSet to automatically create additional replicas when demand increases, and scale back down when demand decreases.
Vertical scaling, also known as scaling up, involves increasing the resources (such as CPU or memory) available to an existing instance of an application. Kubernetes can handle this through the use of a feature called the Horizontal Pod Autoscaler (HPA), which automatically adjusts the number of replicas based on CPU or memory utilization. When the utilization exceeds a specified threshold, the HPA will increase the number of replicas, and when it falls below a certain threshold, it will decrease the number of replicas.

5. **What is a Kubernetes Deployment and how does it differ from a ReplicaSet?**
In Kubernetes, a deployment is a higher-level object that provides declarative updates for replica sets and pods. A deployment is responsible for managing the desired state of a set of pods, ensuring that the current state matches the desired state.
A deployment creates and manages replica sets, which in turn manage pods. A replica set ensures that a specified number of replicas of a pod are running at any given time. If a pod fails or is deleted, the replica set replaces it with a new pod.
In short, deployment is a higher-level object that manages the desired state of a set of pods, while a replica set is a low-level object that manages the scaling and lifecycle of pods. Deployments provide declarative updates, rolling updates, and rollbacks, while replica sets are primarily used for scaling and ensuring the desired number of replicas of a pod are running.

6. **Can you explain the concept of rolling updates in Kubernetes?**
In Kubernetes, a rolling update is a strategy for updating a deployment or a replica set without causing any downtime or interruption to the application. It works as follows:
   - First, Kubernetes creates a new version of the desired deployment or replica set.
   - Next, Kubernetes gradually replaces the old pods with new ones, one at a time, until all the pods have been updated.
   - During the update process, both old and new pods are running simultaneously. This ensures that the application remains available throughout the update process.
   - Once all the new pods are running, Kubernetes deletes the old pods.

   This gradual replacement process is a way to update an application without taking it offline or causing any disruption to users. Rolling updates also provide the ability to roll back to the previous version in case something goes wrong during the update process.
   Overall, rolling updates are a powerful and essential feature of Kubernetes that help keep applications up-to-date and available to users.

7. **How does Kubernetes handle network security and access control?**
Kubernetes has a several built-in features for managing network security and access control. Some of these features are
Network policies: Kubernetes allows administrators to define Network Policies that specify rules for traffic flow within the cluster. These policies can be used to restrict traffic between pods, namespaces, or even entire clusters, based on IP addresses, ports, or other attributes.
Role-Based Access Control (RBAC): Kubernetes supports RBAC, which enables administrators to define granular permissions for users and services based on their roles and responsibilities. This feature allows administrators to control access to Kubernetes resources, including pods, nodes, and services.
Container Network Interface (CNI): Kubernetes supports CNI, which is a plugin-based interface that allows third-party network providers to integrate with the cluster. This feature allows administrators to use their preferred networking solution to provide additional network security and access control.

8. **Can you give an example of how Kubernetes can be used to deploy a highly available application?**

Let's say you have a web application that needs to be highly available, meaning it can't go down if one or more of its components fail. You can use Kubernetes to deploy this application in a highly available manner by doing the following:

- Create a Kubernetes cluster with multiple nodes (virtual or physical machines) that are spread across multiple availability zones or regions.
- Create a Kubernetes Deployment for your application, which specifies how many replicas (copies) of your application should be running at any given time.
- Create a Kubernetes Service for your application, which provides a stable IP address and DNS name for clients to access your application.
- Use a Kubernetes Ingress to route traffic to your application's Service, and configure the Ingress to load-balance traffic across all the replicas of your application.

By following these steps, Kubernetes will automatically monitor your application and ensure that the specified number of replicas are always running, even if one or more nodes fail. Clients will be able to access your application through the stable IP address and DNS name provided by the Service, and the Ingress will distribute traffic across all available replicas to ensure that the application remains highly available.

9. What is namespace is Kubernetes? Which namespace any pod takes if we don't specify any namespace?

Namespace can be recognised as a virtual cluster inside your Kubernetes cluster. We can have multiple namespaces inside a single Kubernetes cluster, and they are all logically isolated from each other. They can help us and our teams with organization, security, and even performance!
There are two types of Kubernetes namespaces: Kubernetes system namespaces and custom namespaces.
If we don't specify a namespace for a pod, it will be created in the default namespace by default. This is the namespace that Kubernetes creates automatically when we set up a cluster, and it is used for objects that do not have a specific namespace specified.
Here are four default namespaces Kubernetes creates automatically

- default
- Kube-system
- Kube-public
- Kube-node-lease

10. How ingress helps in Kubernetes?

Ingress is a Kubernetes resource that provides a way to manage incoming traffic to your cluster. It acts as a layer 7 (application layer) load balancer and provides advanced routing and path-based rules for HTTP and HTTPS traffic.
Here are a few ways that Ingress helps in Kubernetes:
Load balancing: Ingress can be used to distribute traffic to different services in the cluster, based on the URL or hostname specified in the incoming request. This helps to balance the load on the different services and ensure that the traffic is routed to the correct backend service.
Routing: Ingress can route traffic to different services based on the URL path or hostname specified in the request. This makes it easy to manage multiple services running on the same cluster.
Access control: Ingress can be used to restrict access to services based on IP addresses, HTTP headers, or other criteria. This helps to improve the security of the cluster by preventing unauthorized access to the services.
TLS termination: Ingress can terminate SSL/TLS encryption, allowing you to use a single certificate for multiple services and domains.

11. Explain different types of services in Kubernetes?

There are four types of services in Kubernetes:
ClusterIP (default): ClusterIP service is responsible for providing a stable IP address for a set of pods in the cluster. This IP address is only accessible within the cluster, and it allows other services and pods to access the pods that belong to the ClusterIP service.

NodePort: This type of service exposes a set of pods to the outside world. A NodePort service maps a port on each node in the cluster to a specific port on the pod. This service type is used when you need to access a service from outside the cluster or from a different namespace within the same cluster.
LoadBalancer: This type of service is used to expose a set of pods to the outside world through a load balancer. The LoadBalancer service type is used in cloud environments, and it is responsible for automatically creating a cloud load balancer and configuring it to route traffic to the pods in the service.
ExternalName: This type of service maps a service to an external DNS name, allowing you to use the DNS name to access the service instead of the IP address. This service type is useful when you have a service running outside of the cluster, and you want to access it from within the cluster.

12. Can you explain the concept of self-healing in Kubernetes and give examples of how it works?
Self-healing is a feature provided by the Kubernetes open-source system. If a containerized app or an application component fails or goes down, Kubernetes re-deploys it to retain the desired state. Kubernetes provides self-healing by default.
Ex: Suppose we have a web application deployed in Kubernetes with 2 replicas. Each replica runs in its own container. Kubernetes monitors the health of each container by sending periodic requests to the application's endpoints.

If one of the replicas fails, Kubernetes detects it by monitoring the responses to the health check requests. It then terminates the failed container and starts a new one to replace it, ensuring that the total number of replicas is always maintained. The replacement container is created from the same image and configuration as the original, which helps to ensure consistency across replicas.

Kubernetes also supports rolling updates, which allow you to update your application without causing downtime. When you update your application, Kubernetes creates a new set of replicas with the updated code and configuration. It then gradually replaces the old replicas with the new ones, ensuring that the application remains available during the update process.

13. How does Kubernetes handle storage management for containers?
Kubernetes provides various ways to manage storage for containers, including:
volumes, persistent volumes, and storage classes.

- **Volumes:** A volume is a directory accessible to containers in a pod. Volumes are used to store data that needs to persist beyond the lifetime of a container. Kubernetes supports several types of volumes, such as emptyDir, hostPath, configMap, secret, and more. Each type of volume has its own properties and behaviour.
- **Persistent Volumes:** Persistent volumes (PVs) are storage resources provisioned by an administrator that can be used by a pod. A PV is a piece of storage in the cluster that has been provisioned by an administrator. It is not tied to any specific pod, and can be used by any pod that requests it. PVs can be dynamically provisioned by a storage class or statically provisioned by an administrator.
- **Storage Classes:** A storage class is used to define the types of storage that can be dynamically provisioned in the cluster. Storage classes provide a way to abstract the underlying storage infrastructure, making it easier to manage and use storage resources in a consistent way. A storage class defines the provisioner that will be used to provision the storage, along with other parameters like the access mode, reclaim policy, and more.

14. How does the NodePort service work?
NodePort service in Kubernetes allows you to expose a container running in a Kubernetes cluster to the outside world by mapping a specific port of the Kubernetes node to the container's port.

- First, you create a NodePort service by defining it in a Kubernetes manifest file. In this manifest file, you specify the target port on which your container is listening and the port on which you want to expose the service to the outside world.
- When you create the service, Kubernetes assigns a random port in the range of 30000-32767 to the service. This port is the "NodePort" that gives the service its name.
- Kubernetes then creates a mapping between the NodePort and the target port of your container.

- When you want to access your container from outside the Kubernetes cluster, you can use the IP address of any node in the cluster along with the NodePort to access the service. For example, if the NodePort assigned to your service is 32000 and you have a node with IP address 10.0.0.100, you can access the service at http://10.0.0.100:32000.
- The node that you use to access the service will route the traffic to the correct pod based on the mapping that Kubernetes created between the NodePort and the target port of your container.

15. What is a multinode cluster and single-node cluster in Kubernetes?

In Kubernetes, a node is a worker machine that runs containerized applications. A cluster is a group of nodes that work together to run and manage these applications.

A single-node cluster in Kubernetes consists of only one node, which means that all the applications and services are running on the same node. This configuration is useful for testing and development purposes, but it is not recommended for production environments.

On the other hand, a multinode cluster consists of multiple nodes that work together to distribute the workload and provide high availability. In a multinode cluster, if one node fails, the applications can be automatically moved to another node, ensuring that the applications remain available.

16. Difference between create and apply in Kubernetes?

In Kubernetes, "create" and "apply" are two different commands used to manage Kubernetes resources.

"Create" command:

The "create" command is used to create a new Kubernetes resource from a YAML or JSON file. When you create a resource using the "create" command, Kubernetes will create a new resource object based on the specification provided in the file. If a resource with the same name already exists, the "create" command will return an error.

"Apply" command:

The "apply" command is used to create or update a Kubernetes resource based on a YAML or JSON file. When you apply a resource using the "apply" command, Kubernetes will update the existing resource object if it already exists, or create a new resource object if it doesn't exist.

The main difference between the two commands is that "create" always creates a new resource, while "apply" can create or update an existing resource.
