1) Kubernetes?

A) Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. It follows a client-server architecture. Originally developed by Google, it has become the standard for managing containerized workloads and services.

2) Kubernetes benifits?

A) Kubernetes offers several benefits: High Availability, Scalability, efficient resource utilization, Self-Healing.

High Availability::

Kubernetes ensures that your application remains accessible even during server disruptions. If a Pod fails, Kubernetes can create a replacement Pod in the cluster.

Scalability::

Horizontal scaling (adding more instances) and vertical scaling (increasing resources for existing instances) are both supported by Kubernetes.

Self-Healing::

Kubernetes automatically restarts failed containers and replaces unhealthy Pods. This self-healing capability minimizes downtime and ensures application reliability.

Reduced Cloud Costs::

By optimizing resource usage and enabling efficient scaling, Kubernetes can lead to cost savings in cloud deployments

Disaster Recovery::

Disaster recovery as the name suggests is set of practices to ensure that operations are restored after any disruption happens in our deployment system.

Easier Replication::

Replication simply means creating copies of the application so that the application can be scaled

Smart Scheduling::

In Kubernetes, Smart Scheduling is a feature that provides an advanced approach for the scheduling of tasks where we as administrators only have to add new replicas and Kubernetes automatically figures out where these replicas should be running.

3) Explain the concept of Container Orchestration?

A)Container orchestration is the process of automating the management of containerized applications. It involves tasks like deploying, scaling, and managing containers to ensure they run smoothly across different environments.

4) what is pod?

A) Pods are the basic building blocks of applications in Kubernetes and a pod is a logical group of one or more containers that are tightly coupled and share the same resources, such as network namespace, IP address, and storage volumes.

5)How does Kubernetes handle container scaling?

A)Kubernetes handles container scaling through a feature called **Horizontal Pod Autoscaler (HPA)**. Horizontal scaling indicates that more pods are added in response to an increase in load.

6) What is Kubelet?

A)Kubelet is an important component of Kubernetes that manages containers within pods on a node. It registers the node with the control plane and provides resource information. Kubelet keeps an eye on container health.

7) How does Kubernetes manage configuration?

A)Kubernetes employs ConfigMaps and Secrets to manage configuration. ConfigMaps store non-sensitive setup data, while Secrets handles sensitive information like passwords.ConfigMaps have key-value pairs. Sensitive data is securely stored in Secrets, which are applied to the cluster using kubectl. Both

ConfigMaps and Secrets are defined in YAML files and applied to the cluster using kubectl.

8)Describe the role of a Master node in Kubernetes?

A)The Master node in Kubernetes plays a crucial role in managing and controlling the entire Kubernetes cluster. A cluster must have at least one master node; there may be two or more for redundancy. Components of the master node include the API Server, etcd (a database holding the cluster state), Controller Manager, and Scheduler.

9)With Docker I was able to run each component in a separate container – with its own libraries and its own dependencies. Irrespective of what underlying OS they run, all they needed to do was to make sure they had Docker installed on their systems.

10)what are containers?

Containers are completely isolated environments, as in they can have their own processes, and their own network interfaces just like Virtual machines, except that they all share the same OS kernel. Docker utilizes LXC containers.

11) Docker can run any flavor of OS on top of it as long as they are all based on the same kernel – in this case Linux.you wont be able to run a windows based container on a Docker host with Linux OS on it. For that you would require docker on a windows server.

12)virtual machine has its own OS inside it. This overhead causes higher utilization of underlying resources as there are multiple virtual operating systems and kernel running. The virtual machines also consume higher disk space as each VM is heavy and is usually in Giga Bytes in size, were as docker containers are lightweight and are usually in Mega Bytes in size.This allows docker containers to boot up faster, usually in a matter of seconds where as VMs we know takes minutes to boot up as it needs to bootup the entire OS. Docker has less isolation as more resources are shared between containers like the kernel etc. Whereas VMs have complete isolation from each other. Since VMs don't rely on the underlying OS or kernel, you can run different types of OS such as linux based or windows based on the same hypervisor.

13)an image is a read-only template used to create Docker containers. Images are composed of various Layers created using the Dockerfile Instructions. Images contain the application code, libraries, dependencies, and configuration files needed to run the application. Image is a Static file that defined the properties of the Container and its dependencies. You can create your own images or pull pre-built ones from Docker Hub.

14)Containerization is a process of packaging your application together with its dependencies into one package (a container). Such a package can then be run pretty much anywhere. By abstracting the infrastructure, containerization allows you to make our application truly portable and flexible.

15)With Docker, a major portion of work involved in setting up the infrastructure is now in the hands of the developers in the form of a Docker file. This image can now run on any container platform and is guaranteed to run the same way everywhere. So the Ops team now can simply use the image to deploy the application.

16)Each machine in a Kubernetes cluster is called a node. There are two types of node in each Kubernetes cluster: Master node(s): this node hosts the Kubernetes control plane and manages the cluster; Worker node(s): runs your containerized applications.

MASTER NODE:

Control Plane: The control plane manages and controls the entire Kubernetes cluster

API server: API server acts as the front-end for kubernetes. The users, management devices, Command line interfaces all talk to the API server to interact with the kubernetes cluster.

ETCD : ETCD is a distributed reliable key-value store used by kubernetes to store all data used to manage the cluster. ETCD is responsible for implementing locks within the cluster to ensure there are no conflicts between the Masters.

Scheduler: The scheduler assigns pods to nodes based on resource availability and other constraints.

Controller manager: The controller manager manages various controllers that handle cluster-wide functions such as replication, scaling, and managing node health. The controllers are the brain behind orchestration. They are responsible for noticing and responding when nodes, containers or endpoints goes down. The controllers makes decisions to bring up new containers in such cases.

WORKER NODE:

Node: A node, also known as a worker node, is a physical or virtual machine where containers are deployed and run.

Container Runtime : The container runtime, such as Docker, is responsible for pulling container images and running containers on the node

Kubelet : kubelet is the agent that runs on each node in the cluster and communicates with the master node. It manages the containers and ensures they are running and healthy.

Kube-proxy: Kube-proxy manages networking in the Kubernetes cluster by directing traffic between pods and services. **kube-proxy** is primarily responsible for load balancing in kuberntes and service discovery easier.

Additionally, **Ingress Controllers** can also handle load balancing for external traffic coming into the cluster.

17)The master server has the kube-apiserver and that is what makes it a master. Similarly the worker nodes have the kubelet agent that is responsible for interacting with the master to provide health information of the worker node and carry out actions requested by the master on the worker nodes.

18)Kubectl is a command line tool for Kubernetes that allows you to communicate and control Kubernetes clusters. Kubectl works by communicating with the Kubernetes API server. You can use kubectl to create, inspect, update, and delete Kubernetes objects, deploy applications, inspect and manage cluster resources, and view logs.

=> kubectl cluster-info

19)you want these helper containers to live along side with your application container. In that case, you CAN have both of these containers part of the same POD. The two containers can also communicate with each other directly by referring to each other as 'localhost' since they share the same network namespace. Plus they can easily share the same storage space as well.

=> kubectl run nginx --image=nginx :: Is used to create and run a pod with the Nginx container image in a Kubernetes cluster.

20)The kubectl get PODs command helps us see the list of pods in our cluster.

21)Kubernetes uses manifast files as input for the creation of objects such as PODs, Replicas, Deployments, Services etc. All of these follow similar structure. A kubernetes definition file always contains 4 top level fields. The apiVersion, kind, metadata and spec. These are top level or root level properties.

apiVersion: The first one is the apiVersion. This is the version of the kubernetes API we're using to create the object.

Kind : The kind refers to the type of object we are trying to create, which in this case happens to be a POD. So we will set it as Pod.

metadata : The metadata is data about the object like its name, labels etc.

spec : The last section in the configuration file is the specification which is written as spec. Depending on the object we are going to create.

In Kubernetes, spec stands for "specification." It's a key part of Kubernetes resource definitions, such as in pods, deployments, and services. The spec section outlines the desired state of the resource. It tells Kubernetes what the resource should look like, how it should behave. For example, in a deployment YAML, the spec can define the number of replicas, the container images to use, the ports to expose, and more. It's like giving Kubernetes a blueprint for your resources.

22)kubectl describe pod ➔ This will tell you information about the POD, when it was created, what labels are assigned to it, what docker containers are part of it and the events associated with that POD.

23) Looking at our file, we now have two metadata sections – one is for the Replication Controller and another for the POD and we have two spec sections – one for each.

24) Remember that the template and replicas are direct children of the spec section. So they are siblings and must be on the same vertical line.

25) However, there is one major difference between replication controller and replica set. Replica set requires a selector definition. The selector section helps the replicaset identify what pods fall under it.

26)There were pods created BEFORE the creation of the ReplicaSet that match the labels specified in the selector, the replica set will also take THOSE pods into consideration when creating the replicas.

27)There could be 100s of other PODs in the cluster running different application. This is were labelling our PODs during creation comes in handy. We could now provide these labels as a filter for replicaset. Under the selector section we use the matchLabels filter and provide the same label that we used while creating the pods. This way the replicaset knows which pods to monitor.

```
> kubectl create –f replicaset-definition.yml

> kubectl get replicaset

> kubectl delete replicaset myapp-replicaset *Also deletes all underlying PODs

> kubectl replace -f replicaset-definition.yml

> kubectl scale -–replicas=6 –f replicaset-definition.yml

> kubectl scale -–replicas=6 replicaset myapp-replicaset
                                 TYPE       NAME
```

28)The deployment provides us with capabilities to upgrade the underlying instances seamlessly using rolling updates, undo changes, and pause and resume changes to deployments.

➔ To see all the created objects at once run the "kubectl get all" command.

➔ kubectl rollout undo deployment/myapp-deployment

➔kubectl rollout status deployment/myapp-deployment

➔ kubectl rollout history deployment/myapp-deployment

29)There are two types of deployment strategies. One is Recreate strategy and another one is  Rolling Update.

Recreate strategy: It deletes all all the older versions applications at once and creates new versions of application

Rolling update: The Rolling update strategy is were we do not destroy all of them at once. Instead we take down the older version and bring up a newer version one by one. This way the application never goes down and the upgrade is seamless., RollingUpdate is the default Deployment Strategy.

30)The selector is like a matchmaker in Kubernetes, linking your pods to your deployment. Without it, Kubernetes doesn't know which pods your deployment should manage. If you don't specify a selector in your deployment spec, Kubernetes will auto-generate one based on the labels defined in the pod template. However, it's a good practice to explicitly specify the selector to avoid any surprises.

| | |
|---|---|
| create | > kubectl create –f deployment-definition.yml |
| get | > kubectl get deployments |
| update | > kubectl apply –f deployment-definition.yml |
| | >kubectl set image deployment/myapp-deploy nginx=nginx:1.9.1 |
| status | > kubectl rollout status deployment/myapp-deployment |
| | > kubectl rollout history deployment/myapp-deployment |
| rollback | > kubectl rollout undo deployment/myapp-deployment |

31)in Kubernetes the IP address is assigned to a POD. Each POD in kubernetes gets its own internal IP Address.So how is it getting this IP address?

When Kubernetes is initially configured it creates an internal private network with the address 10.244.0.0 and all PODs are attached to it. When you deploy multiple PODs, they all get a separate IP assigned. The PODs can communicate to each other through this IP. But accessing other PODs using this internal IP address MAY not be a good idea as its subject to change when PODs are recreated.

32)This is NOT going to work well when the nodes are part of the same cluster. The PODs have the same IP addresses assigned to them and that will lead to IP conflicts in the network. When a kubernetes cluster is SETUP, kubernetes does NOT automatically setup any kind of networking to handle these issues. As a matter of fact, kubernetes expects US to setup networking to meet certain fundamental requirements.Fortunately, we don't have to set it up ALL on our own as there are multiple pre-built solutions available. Some of them are the cisco ACI networks, Cilium, Big Cloud Fabric, Flannel, Vmware NSX-t and Calico. Depending on the platform you are deploying your Kubernetes cluster on you may use any of these solutions.

33)With the Calico networking setup, it now manages the networks and Ips in my nodes and assigns a different network address for each network in the nodes.This creates a virtual network of all PODs and nodes were they are all assigned a unique IP Address.

34)Services provide a way to expose and access applications running inside pods. A Service acts as an abstraction layer that enables communication between different parts of an application or between different applications.

Service Types:

a. ClusterIP: The default type, which assigns a stable internal IP address to the Service within the cluster. This type allows communication between different pods within the cluster.

b. NodePort: Exposes the Service on a static port on each cluster node. This type allows external access to the Service using the node's IP address and the assigned port.

c. LoadBalancer: Creates an external load balancer that automatically routes traffic to the Service. This type is typically used in cloud environments that support external load balancers. Services distribute incoming traffic across all the pods associated with the Service, providing load balancing functionality.

35)Pull the labels from the pod-definition file and place it under the selector section. This links the service to the pod.

36) TargetPort: This is the actual port on which your application is running inside the container.

Port: The port is were the service is exposed.

NodePort: Assigns a port within the range of (30000-32767) to expose the application to the internet from where end-users can access it.

37)Kubernetes has built-in integration with supported cloud platforms.

38)One way to achieve this in my current VirtualBox setup is to create a new VM for Load Balancer purpose and install and configure a suitable load balancer on it like HAProxy or NGINX etc. The load balancer would then re-route traffic to underlying nodes and then through to PODs to serve the users. However, if I was on a supported cloud platform like Google Cloud, I could leverage the native load balancer to do that configuration for me. Kubernetes has support for getting that done for us automatically. All you need to do is set the Service Type for the front end services to LoadBalancer. Remember this only, works with supported cloud platforms.

39)Container orchestration brings several benefits to application deployment and management, including improved scalability, fault tolerance, resource utilization, and simplified deployment processes.

40) In summary, Docker Swarm is a simpler and more lightweight solution suitable for smaller deployments,while Kubernetes offers a more robust and scalable platform for managing large-scale containerized applications.

41)Pod Scheduling: Pods are scheduled to run on specific nodes in the cluster based on resource requirements and node availability. The Kubernetes scheduler ensures that pods are evenly distributed across nodes.

42)If an issue is detected during a rolling update,Kubernetes can automatically halt the update process and roll back to the previous version.

43) Horizontal Pod Autoscaler (HPA) is a Kubernetes feature that automatically adjusts the replica count based on CPU utilization or custom metrics.With HPA, you can set minimum and maximum replica counts and define thresholds for scaling up or down based on resource usage.

44) Explain the concept of Ingress in Kubernetes?

Ingress in Kubernetes is a resource that manages external access to services within a Kubernetes cluster, typically via HTTP and HTTPS. It acts as a gateway, providing a single entry point for external traffic and routing it to the appropriate services based on defined rules.

Key Features of Ingress:

Routing: Ingress allows you to define rules for routing traffic to different services based on the request's host, path, or other attributes.

Load Balancing: It can distribute incoming traffic across multiple backend services, providing load balancing.

SSL/TLS Termination: Ingress can handle SSL/TLS termination, managing certificates and decrypting HTTPS traffic before passing it to the backend services.

How Ingress Works:

Ingress Resource: You define an Ingress resource in your Kubernetes cluster, specifying the routing rules.

Ingress Controller: An Ingress controller is responsible for fulfilling the Ingress resource. It typically uses a load balancer to manage traffic according to the defined rules. Popular Ingress controllers include NGINX, Traefik, and HAProxy.

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: example-ingress
```

45)What is a ConfigMap?

A ConfigMap is an API object in Kubernetes that is primarily used to store non-confidential data. ConfigMaps are a way for Kubernetes to inject configuration data into application pods, making it easier to manage and update configuration settings.

46)What is a Namespace in Kubernetes?

Namespaces permit Kubernetes clusters to be organized into virtual sub-clusters, which is useful in situations where a cluster is utilized by several teams or projects.

47)Explain the use of Labels and Selectors in Kubernetes.

Labels and Selectors are essential sections in Kubernetes configuration files for deployments and services due to how they link Kubernetes services to pods. Labels are key-value pairs that identify pods distinctly; the deployment assigns these labels and uses them as a starting point for the pod prior to its creation, and the Selector matches these labels. Labels and selectors combine to create connections between deployments, pods, and services in Kubernetes.

48) What is a Persistent Volume (PV) in Kubernetes?

A Persistent Volume (PV) in Kubernetes is an object that allows pods to access storage from a defined device. This device is usually described via a Kubernetes StorageClass.

```
apiVersion: v1

kind: PersistentVolume
```

49)What advantages does Kubernetes have?

Kubernetes has the following advantages:

Container Orchestration

Automated Load Balancing

Auto Scaling

Rolling Update & Rollbacks

Service Discovery and Load Balancing

Storage Orchestration

Self-Healing

Secrets and Configuration Management

Multi-Cloud and Hybrid Cloud Support

Role-Based Access Control (RBAC)

Pods and Multi-Container Support

Monitoring and Logging

50)How does Kubernetes handle security and access control?

Using robust access restrictions and encryption techniques, like Kubernetes Secrets or external key management systems, can help safeguard sensitive data kept within your cluster. To prevent unwanted access, data must be encrypted as it is in transit and at rest.

51)How does Kubernetes manage storage orchestration?

The Container Storage Interface (CSI) is the standard to establish device-independent relationships across block and file storage systems and containerized workloads.

1. Kubernetes => An open-source container orchestration tool/system used for automating tasks like management, monitoring, scaling, and deployment of containerized applications.

2. Orchestration => orchestration refers to the automated coordination and management of complex workflows and tasks across various systems and environments.

3. Architecture => Understanding Kubernetes components: master node, worker nodes, etcd, API server, controller manager, and scheduler.

4. Pods => The basic unit in Kubernetes, containing one or more containers.

5. Services => Exposing pods to the network. Types: ClusterIP, NodePort, Load Balancer, and ExternalName.

6. Replication Controllers => Ensuring a specified number of replicas of a pod are running.

7. Deployments => Managing rolling updates and rollbacks.

8. ConfigMaps and Secrets => Managing configuration data and sensitive information.

9.Persistent Volumes => Managing storage for stateful applications.

10.Ingress Controllers => Routing external traffic to services within the cluster.

11. Helm => Package manager for Kubernetes.

12. Security => RBAC, Network Policies, and Pod Security Policies.

13. Monitoring and Logging => Prometheus, Grafana, and ELK stack.