Deep Chavan

T11-15

Assignment No. 10

Aim: To complete study of Kuberenetes.

Theory:

- What are the various Kubernetes services running on nodes? Describe the role of each service.

Kubernetes is a complex container orchestration platform, and it relies on several services and components that run on nodes to manage and control containerized applications. These services work together to provide the functionality required for deploying, scaling, and maintaining applications. Here are some of the key Kubernetes services running on nodes, along with their roles:

Kubelet:

Role: The Kubelet is an essential component that runs on each node in the cluster. It is responsible for ensuring that containers are running in a Pod. It takes Pod specifications (from the API server) and makes sure the containers described in the Pod are running and healthy on the node.

Container Runtime (e.g., Docker, containerd, or CRI-O):

Role: The container runtime is responsible for pulling container images, running containers, and managing their lifecycle. It's the software that actually runs containerson the node.

Kube Proxy:

Role: Kube Proxy is responsible for network connectivity in the cluster. It maintains network rules on each node to route traffic to the appropriate pods based on services and labels. It helps provide load balancing and service discovery.

CNI Plugins (Container Network Interface):

Role: CNI plugins are responsible for setting up and managing the network connectivity for containers in the pod. These plugins configure the network interfaces, IP addresses, and routes for containers so they can communicate with each other and with external networks.Node Status and Heartbeat (NodeStatus and NodeHeartbeat):

Role: These services are responsible for sending the current status and health of the node to the control plane. They provide information about the node's available resources, conditions, and any issues it might be facing.

Docker Daemon (if using Docker):

Role: If you're using Docker as the container runtime, the Docker Daemon manages container images, storage, and the execution of containers.

Containerd (if using Containerd):

Role: Containerd is an alternative container runtime that performs similar functions toDocker. It is used when you opt for a more lightweight runtime compared to Docker.

Kubelet Registrar and Garbage Collector:

Role: These components help with registering nodes with the control plane and ensuring that nodes and their resources are cleaned up and properly managed.

Node OS (Operating System):

Role: The underlying node OS is responsible for providing the necessary environment for running containers. It includes the kernel, system libraries, and utilities required for containers to function.

Systemd or Init System:

Role: The init system (e.g., systemd) on the node is responsible for managing system processes and ensuring that the Kubernetes services are started at boot time and restarted if they fail.

- What is Pod Distribution Budget?

A Pod Distribution Budget is a concept in Kubernetes that allows you to specify constraints and requirements on how pods should be distributed across nodes in a cluster. It is used to control the distribution of pods to ensure that they are evenly spread across nodes or follow specific rules for placement. This is particularly useful for enhancing high availability and fault tolerance, improving resource utilization, and ensuring compliance with policies or regulations.

Pod Distribution Budgets are part of the Pod Topology Spread Constraints feature,which was introduced in Kubernetes to provide more control over how pods are scheduled on nodes.

Here are the key components of a Pod Distribution Budget:

Topology Key: A topology key is a label or field key that is used to determine the distribution of pods across nodes. For example, it could be the availability zone or region of a node.

Max Skew: Max Skew is a numerical value that represents the maximum allowed imbalance in the distribution of pods. For example, if the Max Skew is set to 1, it means that the difference in the number of pods across nodes should not exceed 1.

Topology Spread Constraints: These are rules that specify how pods should be distributed based on the topology key. For instance, you can define constraints to ensure that pods are distributed evenly across nodes in different availability zones.

Pod Distribution Budgets are created by defining a CustomResourceDefinition (CRD) object in Kubernetes. You can then reference this object when creating or updating Deployments, StatefulSets, or other types of workload controllers. The Pod Distribution Budget constraints are checked during the scheduling process, ensuring that pods are placed on nodes in compliance with the defined rules.

By using Pod Distribution Budgets, you can improve the resilience and stability of your applications by avoiding overloading specific nodes and ensuring that your workloads are evenly distributed across your cluster, which is essential for achieving a balanced and efficient Kubernetes cluster.

- What is the role of loadbalance in Kubernetes?

Load balancing in Kubernetes plays a crucial role in ensuring the availability, scalability, and reliability of applications and services running within the cluster. Here'san
overview of the role of load balancing in Kubernetes:

Service Accessibility: Kubernetes abstracts the underlying infrastructure, which means that pods and services can be distributed across different nodes in the cluster. Load

balancing is used to ensure that these services remain accessible to clients regardless of the specific node or pod where they are running.

Service Discovery: Kubernetes Services allow you to define a stable DNS name or IP address for a set of pods. Load balancers route traffic to the appropriate pods associated with a service based on selectors, labels, and other criteria, making it easy for clients to discover and connect to services without needing to know the specific details of pod locations.

Scaling: Kubernetes allows for automatic scaling of pods based on CPU or other resource usage metrics. Load balancers distribute traffic across these dynamically changing instances, ensuring that traffic is evenly distributed even as pods are added or removed in response to demand.

High Availability: Load balancers provide high availability by distributing traffic across multiple replicas of an application. If one pod or node fails, the load balancer can route traffic to healthy instances, minimizing downtime.

Security: Load balancers can serve as a point of entry for traffic into the cluster, providing a layer of security by controlling which traffic is allowed and which is not. They can also be used to terminate SSL/TLS encryption, ofloading the decryption process from the application.

Traffic Splitting: Load balancers support advanced traffic management features like A/B testing and canary deployments. They allow you to route a percentage of traffic to one version of an application and a different percentage to another version for testing or
gradual rollouts.

Global Load Balancing: For multi-cloud or multi-region deployments, Kubernetes can utilize global load balancers to route traffic to the nearest or most available data center or cloud region. This improves latency and redundancy.

Request Routing and Path-Based Routing: Load balancers can route traffic based on different criteria, such as the path in the URL. This enables microservices architectures where different paths are handled by different services.

Kubernetes itself does not provide a load balancer but integrates with various load balancing solutions, such as external cloud load balancers, NodePort services, ClusterIP services, and Ingress controllers. The choice of load balancer depends on the specific requirements and the underlying infrastructure used in your Kubernetes cluster.

Conclusion:

In this assignment we learnt what exactly are kubernetes.