Animesh Parab T2-T21 88

ASSIGNMENT-5

AIM: To understand the Kubernetes Cluster Architecture.

LO MAPPED: LO1 , LO3

THEORY:

Q.1 What are the various Kubernetes services running on nodes? Describe the role of each service.

In a Kubernetes cluster, there are several essential services that run on nodes. These services are critical for the proper functioning of the cluster. Below, I'll describe the role of each service in detail:

kubelet:

The kubelet is responsible for managing containers on a node. It ensures that the containers in a Pod are running and healthy. It communicates with the control plane to receive Pod specifications and takes actions to make sure the containers match the desired state. For example, if a Pod specification indicates that it should run three containers, the kubelet ensures that those containers are up and running. If a container fails, the kubelet restarts it.

Example: Let's say you have a Pod with three containers, and one of them crashes due to a software issue. The kubelet will detect the failure and restart the failed container to maintain the desired state.

kube-proxy:

Kube-proxy is responsible for managing network connectivity to and from Pods. It maintains network rules on the host to enable communication between Pods and external networks. It sets up routes, handles load balancing, and ensures that network traffic is properly directed to the correct Pod. **Example:** If you have a service in your cluster that needs to load balance traffic to a set of Pods, kube-proxy manages this load balancing by configuring network rules and routes, directing traffic to the appropriate Pods.

Container Runtime:

The container runtime is responsible for running containers within Pods. Kubernetes supports various container runtimes, such as Docker, containerd, and CRI-O. These runtimes are responsible for pulling container images, creating containers, and managing their lifecycle.

Example: If you define a Pod that runs a Docker container with a specific image, the container runtime (e.g., Docker) pulls the image from a container registry and runs the container as specified.

cAdvisor (Container Advisor):

cAdvisor is responsible for collecting and exposing resource usage and performance data for containers. It provides valuable information about CPU, memory, network, and disk usage of running containers.
Example: You can use cAdvisor to monitor the resource consumption of your containers. For instance, it can help you identify a container that is consuming an unusually high amount of CPU or memory, indicating a potential performance issue.
Node Problem Detector (Optional):
The Node Problem Detector is responsible for detecting and reporting hardware and system failures on the node. It helps in identifying and isolating issues with nodes, such as hardware errors, kernel panics, or out-of-memory conditions.
Example: If a node experiences a hardware issue, such as a failing disk drive, the Node Problem Detector can detect this problem and report it, allowing administrators to take action and potentially drain the node to prevent further issues.
Device Plugins (Optional):
Device plugins are used to expose and manage specialized hardware resources on the node, such as GPUs, FPGAs, or hardware accelerators. They enable Pods to use these resources when required.
Example: If you have GPUs on your nodes and want to run machine learning workloads that require GPU acceleration, you can use a GPU device plugin toexpose these GPUs to your Pods. Pods that need GPU resources can request them in their specifications.
OS Services (e.g., SSH, NTP):
These services, including SSH for remote access and NTP for time synchronization, are essential for maintaining the health and reliability of the node. SSH provides administrative access for troubleshooting and maintenance, while NTP ensures the node's clock is synchronized with the cluster, preventing time-related issues.
Example: You can use SSH to log in to a node for troubleshooting or updates. NTP ensures that all nodes in the cluster have synchronized clocks, which is crucial for maintaining consistency in distributed systems.
Kubelet Container:
The kubelet itself runs in its own container on the node. It is responsible for interacting with the container runtime, managing container logs, and performing garbage collection to reclaim disk space from unused container images.
Example: If you examine a running node, you'll find the kubelet running in its own container. It manages container-related tasks on the node, such as cleaning up old container images to free up storage space.
These roles collectively ensure the smooth operation of a Kubernetes node and the containers within it, making it possible to run and manage containerized applications in a distributed environment.

Q.2 What is Pod Disruption Budget (PDB)?

A **Pod Disruption Budget (PDB**) is a Kubernetes resource that allows you to control the disruption or eviction of Pods during voluntary disruptions (e.g., maintenance) and involuntary disruptions (e.g., hardware failures). PDBs define the minimum availability requirements for Pods in a set of related Pods, such as those belonging to a Deployment or StatefulSet. They ensure that a certain number of Pods are available at all times, helping to maintain the stability and availability of your applications in a Kubernetes cluster.
Here's a detailed explanation of PDBs and an example to illustrate their use:
Components of a Pod Disruption Budget (PDB):
minAvailable: This field specifies the minimum number of Pods that must be kept running in the group (e.g., a Deployment or StatefulSet). This ensures that a minimum number of replicas remain available during disruptions.
maxUnavailable: This field specifies the maximum number of Pods that can be unavailable during disruptions. It's complementary to minAvailable. You can choose to define one or the other, but not both. It provides a way to limit the maximum unavailability of Pods.
selector: PDBs are associated with Pods using label selectors. The selector is used to match Pods in the group that the PDB applies to.
Use Cases for PDBs:
Rolling Updates: When performing rolling updates of applications using Deployments or StatefulSets, PDBs can be used to ensure that a certain number of Pods remain available during the update process, preventing unintended disruptions to the application.
Node Drains: When nodes need maintenance or are being drained, PDBs can prevent the simultaneous eviction of too many Pods, ensuring that the application maintains its desired level of availability.
High Availability: PDBs can be used to enforce high availability requirements for critical components of an application. For example, a database cluster may require a certain number of replicas to be available at all times to prevent data loss.
Example of a Pod Disruption Budget:
Let's say you have a Deployment managing a web application with a replica count of 5. You want to ensure that at least 3 replicas of the web application are available at all times during updates or node maintenance. Here's how you would define a PDB for this use case:

```
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
name: web-app-pdb

spec:
minAvailable: 3
selector:
```

matchLabels:
app: web-app
In this example:
minAvailable: 3 specifies that at least 3 replicas of Pods with the label app: web-app must remain available during disruptions.
selector specifies that this PDB applies to Pods with the label app: web-app.
With this PDB in place, if you perform a rolling update of the Deployment or if nodes need maintenance, Kubernetes will ensure that at least 3 Pods of the web-app Deployment remain operational during these events, thereby meeting the specified availability requirements.
PDBs are a powerful tool for maintaining application stability and availability in Kubernetes clusters, particularly when handling planned or unplanned disruptions to your workloads.

Q.3 What is the role of Load Balance in Kubernetes?

In Kubernetes, a Load Balancer is a critical component that helps distribute network traffic evenly across a set of Pods or Services. Load balancing is essential for ensuring high availability, scaling applications, and maintaining stable network connections. Here's a detailed explanation of the role of Load Balancers in Kubernetes, along with an example:
Role of Load Balancers in Kubernetes:
Distributing Traffic: Load Balancers evenly distribute incoming network traffic across multiple Pods or Services. This ensures that no single Pod or Service becomes overwhelmed, improving the responsiveness and availability of the application.
High Availability: Load Balancers are typically configured with health checks to monitor the status of Pods or Services. If a Pod or Service becomes unhealthy or unresponsive, the Load Balancer can automatically route traffic away from it, ensuring the application remains available.
Scaling: As your application grows and you need to add more instances (Pods) to handle increased traffic, Load Balancers can seamlessly adapt to include these new instances in the traffic distribution. This makes it easier to scale your application horizontally.
Session Persistence: Some Load Balancers support session persistence or sticky sessions, which ensure that requests from the same client are consistently routed to the same backend Pod. This is useful for stateful applications that rely on session data.
External Access: Load Balancers often act as a point of entry for external traffic into your cluster. They can route traffic to the appropriate Services within the cluster based on the configuration and rules you define.
Types of Load Balancers in Kubernetes:
Service Type: LoadBalancer: Kubernetes provides a native LoadBalancer Service type. When you define a Service of type LoadBalancer, the Kubernetes cluster provisions an external Load Balancer, typically

provided by the cloud provider, to distribute traffic to the Service. For example:

```
apiVersion: v1
kind: Service
metadata:
name: my-service
spec:
selector:
app: my-app
ports:
- protocol: TCP
port: 80
targetPort: 9376
type: LoadBalancer
```

Ingress Controllers: Ingress controllers, such as Nginx Ingress or HAProxy, are used to manage external access to Services within a cluster. Ingress controllers
provide more advanced routing and traffic management capabilities than the basic LoadBalancer Service type.

Example of Load Balancer in Kubernetes:

Let's say you have a web application deployed as a set of Pods and you want to make it accessible to external users. You can create a LoadBalancer Service to achieve this:

```
apiVersion: v1
kind: Service
metadata:
name: web-service
spec:
selector:
app: web-app
ports:
- protocol: TCP
port: 80
targetPort: 8080
type: LoadBalancer
```

In this example:

metadata.name is the name of the Service.

spec.selector specifies the Pods to which the traffic should be load balanced.

spec.ports define the ports to which the Load Balancer should forward traffic.

type: LoadBalancer indicates that you want to provision an external Load Balancer for this Service.

Once this configuration is applied, Kubernetes (or the cloud provider) will provision an external Load Balancer and assign it an IP address. Users can then access your web application using this IP address, and the Load

Balancer will distribute incoming requests across the Pods running your web application.
Load Balancers are a fundamental component for ensuring the availability, scalability, and external accessibility of applications in a Kubernetes cluster. They play a crucial role in maintaining a stable and responsive environment for your applications.

CONCLUSION: Hence,In this assignment we learned what is the Kubernetes Cluster Architecture.