|  |  |
| --- | --- |
| **INGRESS** :- In k8s, an Ingress is an API object that defines rules for routing external traffic to internal services within a Kubernetes cluster. It acts as a layer 7 (application layer) load balancer that can route traffic based on the host name or path in the URL.  An Ingress resource provides a way to configure the HTTP(S) load balancing for a Kubernetes service. It defines a set of routing rules that map incoming requests to the appropriate backend service. This allows you to expose multiple services on a single IP address and port, which can be useful for applications that have multiple components or microservices.  Overall, Ingress is a powerful tool for managing external traffic to Kubernetes services, and it helps to simplify the deployment and management of complex applications within a Kubernetes cluster.  Ingress in Kubernetes supports different types of routing, including path-based routing, host-based routing, and TLS termination. |  |

|  |
| --- |
| **Q :-** can we use single external load balancer so that we can access multiple micro services?  **ANS :-** Yes, you can use a single external load balancer to access multiple microservices in Kubernetes. This can be achieved by using Kubernetes Ingress, which provides a way to manage incoming traffic and route it to different microservices based on the URL path or hostname.  With Ingress, you can define a set of routing rules that map incoming requests to the appropriate backend service, using a single IP address and port. This allows you to expose multiple microservices on a single external load balancer, making it easier to manage and scale your applications.  To achieve this, you would need to deploy an Ingress controller within your Kubernetes cluster, which would be responsible for implementing the routing rules defined in the Ingress resource. You would also need to define an Ingress resource that specifies the routing rules for your microservices.  Overall, using a single external load balancer with Ingress can simplify the management and scaling of your microservices architecture, by providing a single point of entry for incoming traffic and allowing you to easily route requests to the appropriate backend service.  Top of Form  Bottom of Form |

**RBAC** (Role-Based Access Control) is a security mechanism used in k8s to manage user access to resources within the cluster. With RBAC, you can define roles and permissions for different users or groups of users, allowing you to control who has access to what resources in the cluster.

RBAC in Kubernetes works by defining four different types of objects:---

**Role :** Role is a set of permissions that define what actions a user or group of users can perform within a specific namespace. Role Works at namespace level.

**ClusterRole :** A ClusterRole is a set of permissions that define what actions a user or group of users can perform across the entire cluster. ClusterRole Works at cluster scope. There is no namespace boundary.

**RoleBinding :** A RoleBinding is a mapping between a Role and a user or group of users. To map role with subjects( Users/Groups/ServiceAccounts ) we will use rolebinding.

**ClusterRoleBinding :** A ClusterRoleBinding is a mapping between a ClusterRole and a user or group of users.

By defining Roles and ClusterRoles, you can define what actions users are allowed to perform within a namespace or across the entire cluster. By defining RoleBindings and ClusterRoleBindings, you can map these permissions to specific users or groups of users.

For example, you can define a Role that allows a user to create and manage pods within a specific namespace, and then create a RoleBinding that maps that Role to the user. Similarly, you can define a ClusterRole that allows a user to manage all pods across the entire cluster, and then create a ClusterRoleBinding that maps that ClusterRole to the user.

RBAC in Kubernetes provides a powerful and flexible way to manage user access to resources within the cluster. By carefully defining Roles and ClusterRoles, and mapping them to specific users or groups of users using RoleBindings and ClusterRoleBindings, you can ensure that users have access only to the resources they need to do their jobs effectively.

In Kubernetes, there are different types of users that may need access to the Kubernetes cluster, including:

**Cluster administrators:** Cluster administrators are responsible for managing the entire Kubernetes cluster, including setting up and configuring the cluster, managing cluster resources, and monitoring the health of the cluster. Cluster administrators have full access to all Kubernetes resources and can perform any operation on the cluster.

**Namespace administrators:** Namespace administrators are responsible for managing a specific namespace within the Kubernetes cluster. They have the ability to create and manage resources within the namespace, but they do not have access to resources outside the namespace.

**Developers:** Developers are responsible for deploying and managing applications within the Kubernetes cluster. They have access to the resources necessary to deploy and manage their applications, but they do not have access to the underlying infrastructure or the ability to modify cluster-level resources.

**End-users:** End-users are the users who interact with the applications deployed on the Kubernetes cluster. They do not have access to the Kubernetes resources directly, but they interact with the applications deployed on the cluster through APIs, web interfaces, or other means.

It's important to carefully manage user access to the Kubernetes cluster to ensure the security and stability of the cluster. Access control can be managed using Kubernetes RBAC (Role-Based Access Control), which allows you to define roles and permissions for different users or groups of users. By carefully managing user access and permissions, you can ensure that the right users have the right level of access to the Kubernetes resources they need to do their jobs effectively.

|  |
| --- |
|  |

**Helm** is a package manager for Kubernetes, which is an open-source container orchestration system. It helps simplify the process of deploying and managing applications on a Kubernetes cluster by providing a way to define, install, and upgrade applications and their dependencies as packages, known as "charts."

Helm is a package manager for Kubernetes that simplifies the deployment and management of applications on Kubernetes clusters. It allows you to package your Kubernetes manifests, also known as charts, into a single unit that can be versioned, shared, and deployed using a single command.

A **Helm chart** is a collection of YAML files that describe a set of Kubernetes resources, such as deployments, services, and config maps, that are required to deploy and manage an application. Helm charts can be customized using templates, which allow you to parameterize the chart and provide configuration options.

With Helm, you can package your Kubernetes application as a chart, which is a collection of files that describe your application's Kubernetes resources, such as pods, services, and deployments. Helm charts can be shared and distributed, making it easy for teams to deploy and manage applications consistently across different environments.

Helm provides a command-line interface (CLI) tool that you can use to search, install, update, and manage charts, as well as customize chart configurations using template files. It also supports versioning, rollback, and dependency management, making it easier to manage complex Kubernetes applications.

There are several benefits of using Helm as a package manager for Kubernetes:-----------

Simplified application deployment: Helm allows you to package all the Kubernetes resources required for an application into a single, easy-to-manage chart. This makes it easier to deploy and manage applications on Kubernetes clusters.

Version control: Helm charts are versioned, making it easier to track changes and revert to previous versions if needed. This is especially useful when managing complex applications with many dependencies.

Reusability: Helm charts can be shared and reused across teams and organizations. This makes it easier to standardize and streamline the deployment of applications, reducing duplication of effort and improving productivity.

Customization: Helm charts can be customized using templates, which allows you to parameterize the chart and provide configuration options. This makes it easier to deploy the same application with different configurations or settings.

Ecosystem: Helm has a large and growing ecosystem of charts that cover a wide range of popular applications, such as databases, web servers, and monitoring tools. This makes it easier to find and use existing charts instead of having to create them from scratch.

Rollbacks: Helm supports rollbacks, which allow you to revert to a previous version of a chart if there are issues with a new version. This reduces the risk of downtime and makes it easier to recover from issues.