**RESOURCES IN K8S**

**What is resource type?**

Resource Type is a kind of object or entity that the kubernetes API manages within the cluster.

These types represent the fundamental building blocks of your applications and infrastructure within a Kubernetes cluster.

**Why resource type?**

* **Provide a structured way to interact with the Kubernetes API:**

Each resource type has a defined schema and API endpoint, allowing users and tools to consistently create, read, update, and delete objects in the cluster.

* **Enable declarative configuration:**

You define the desired state of your application and infrastructure using these resource types in YAML or JSON manifests, and Kubernetes works to achieve and maintain that state.

* **Facilitate automation and orchestration:**

By representing different components as distinct resource types, Kubernetes can effectively manage and orchestrate complex applications, handling scaling, self-healing, and other operational tasks.

* **Allow for extensibility:**

Kubernetes allows for the creation of Custom Resource Definitions (CRDs), which enable users to define their own resource types to extend Kubernetes' capabilities for specific use cases or applications.

**Resources in k8s are classified based on their scope**

1. Cluster scoped resources
2. Namespace scoped resources
3. **Cluster scoped resources:**

Cluster scoped resources are objects that exist at the cluster level. They are accessible across the entire Kubernetes cluster.

**Why cluster scoped resources?**

* **Global Configuration and Management:**

They enable the definition and management of resources that are relevant to the entire cluster.

* **Cross-Namespace Functionality:**

Certain functionalities require a scope beyond individual namespaces to function correctly.

* **Infrastructure Management:**

Resources like Nodes and Persistent Volumes are fundamental to the cluster's infrastructure and are inherently cluster-scoped.

* **Security and Governance**

Administrators can enforce global policies and rules using cluster-scoped resources to ensure compliance and security.

**Cluster-scoped resources are:**

* **Node:**

It is a physical or virtual machine that runs containerized applications as part of a Kubernetes cluster.

Each node is managed by the control plane and contains the services necessary to run the pods.

**Why node?**

Nodes deploy and manage containers, provides resources like CPU, memory, and storage, scales applications across many machines, ensures availability, fault tolerance, and efficient scheduling.

* **PersistentVolume (PV):**

It is a storage like disk.

It is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using storageclasses.

**Why PV?**

PVs provide persistent storage for applications, allowing data to survive the lifecycle of pods, which is essential for stateful applications like databases and message queues that need to retain data.

* **Namespace:**

namespaces are the way of dividing the cluster resources between multiple users.

**Why namespace?**

* **Resource Isolation:** They provide logical separation of resources, ensuring that different applications or teams do not interfere with each other within the same cluster.
* **Access Control:**Namespaces enable fine-grained access controls, allowing administrators to define permissions and policies specific to each namespace.
* **Environment Segregation**: They facilitate the creation of separate environments within a single cluster, improving organization and management.
* **Efficient Resource Management:** Namespaces allow for better resource allocation preventing any single application from consuming excessive resources at the expense of others.
* **ClusterRole:**

Cluster role is a role used to grant access to resources across all namespaces in the cluster.

**Why ClusterRole?**

* It is used to give permissions across the whole cluster.
* It ensures that users, services, or applications only have the permissions they need.
* Helps prevent unauthorized access or accidental changes.
* **CustomResourceDefinition(CRD):**

It is a mechanism that allows users to extend the Kubernetes API by defining their own resource types.

These custom resources behave like built-in Kubernetes objects but are user-defined.

**Why CRD?**

* It extends kubernetes functionality, that is it lets you create custom resources.
* Custom resources are often paired with controllers that automate tasks like scaling, backups, or failover.
* CRDs allow teams to define custom objects in a consistent way that works across development, testing, and production clusters.
* **StorageClass**

It is like a blueprint for dynamic storage in Kubernetes. It will define how PVs should be created when a pvc asks for storage.

Instead of creating pvc’s manually the storage class tells kubernetes to talk to the underlying cloud.

**Why StorageClasses?**

They allow administrators to abstract the underlying storage infrastructure, enabling developers to simply request a specific type of storage without needing to know the technical details of how it's provisioned.

1. **Namespace scoped resources:**

Namespace scoped resources are the objects that resides within a specific Namespace and are logically isolated from resources in other Namespaces.

**Why?**

* They are used to logically partition a single cluster into multiple virtual clusters.
* This helps different teams, projects, or applications share a cluster without their resources interfering with one another.
* Namespaces provide a crucial layer of isolation, which simplifies resource management, avoids naming conflicts, and allows administrators to apply separate security policies and resource quotas to each logical partition.

**Namespace scoped resources are:**

* **Pod**

It is a smallest deployable unit, that you can create and manage in Kubernetes.

**Why Pod?**

* Pods are used to manage and schedule containers as a single, cohesive unit.
* They provide a shared environment for their containers which includes shared network, shared storage.
* **Deployment**

It is a resource that provides a declarative way to manage and update a set of identical Pods.

**Why deployment?**

* A Deployment ensures that the desired number of Pod replicas are always running and available.
* It also provides powerful features like rolling updates, which allow you to update your application without any downtime, and rollbacks, which let you quickly revert to a previous version if an update goes wrong.
* **Replicaset**

It is a Kubernetes controller that ensures a specified number of identical Pod replicas for an application are always running in a cluster.

**Why Replicaset?**

* It is used to maintain the desired state of Pods.
* If a Pod fails, is deleted, or crashes for any reason, the ReplicaSet will automatically create a new one to replace it, ensuring the application remains available.
* **Configmap**

A ConfigMap in Kubernetes is an API object that stores non-confidential configuration data as key-value pairs.

**Why Configmap?**

ConfigMap is used to decouple configuration from your application code and container image. This means you don't need to rebuild your container image every time a configuration value changes. Instead, you can simply update the ConfigMap, and Kubernetes will apply the changes to the running pods.

* **Secret**

It is a secure way to store and manage sensitive information like passwords, API keys, and TLS certificates.

**Why Secret?**

* Secret is used to protect sensitive data and prevent it from being accidentally exposed. Instead of hard-coding credentials directly into your application's source code or a container image, you can store them in a Secret.
* Kubernetes then injects this information into your Pods as either a mounted file or an environment variable.
* **PersistentVolumeClaim (PVC)**

It is a request for storage made by a Pod. It's used by developers and applications to ask for a specific amount and type of storage without needing to know the technical details of the underlying storage infrastructure.

**Why PVC?**

* PVC is used to abstract away the complexities of storage management.
* Instead of a developer having to manually provision a physical disk or cloud-based storage volume, they simply create a PVC that requests what they need.
* Kubernetes then automatically finds an available storage resource that matches the claim and binds it to the Pod.
* **Service**

A Service is a way to expose a set of Pods as a single, consistent network endpoint.

**Why service?**

The primary reason for using a Service is to solve the problem of ephemeral Pods. Since Pods are designed to be temporary and can be replaced at any time their IP addresses are not reliable for communication. A Service gives you a stable IP address and DNS name for your application.