# Cluster scoped resources

* They exist at the cluster level and are not bound to any namespace.

1. **Node**: A worker machine (VM/physical) in the cluster.

Why?

* Nodes belong to the cluster as a whole, not a namespace. Pods are scheduled onto nodes, so it must be global.

1. **PersistentVolume (PV)**: A piece of storage provisioned for cluster use.

Why?

* Storage is a physical/cluster-wide resource, not tied to a namespace. PVCs (namespace-scoped) claim PVs.

1. **Namespace**: A logical partition for grouping resources.

Why?

* You can’t put a namespace *inside* another namespace. Namespaces define the scope for other objects.

1. **ClusterRole**: A set of permissions (verbs on resources) across the cluster.

Why?

* Needed for granting access cluster-wide or to non-namespaced resources.

1. **ClusterRoleBinding**: Connects ClusterRole with users, groups, or service accounts.

Why?

* Ensures access is applied globally, not just inside one namespace.

1. **CustomResourceDefinition (CRD)**: Let’s you define your own resource types.

Why?

* Once defined, the new resource type is available to all namespaces in the cluster.

1. **APIService**: Registers an extension API server with the Kubernetes API.

Why?

* It changes the cluster API itself, so must be cluster-wide.

1. **CertificateSigningRequest (CSR)**: Request object for certificates (TLS).

Why?

* Certificates are issued at the cluster level, often for kubelets/nodes.

1. **IngressClass**: Defines how Ingress objects should be implemented (e.g., which controller to use).

Why?

* Ingress controllers run cluster-wide and must know how to treat all Ingress resources.

1. **Storage Class**: Defines dynamic provisioning of PVs (storage backend, parameters).

Why?

* Storage provisioning applies cluster-wide, across namespaces.

1. **VolumeAttachment**: Tracks the attachment of a volume to a node.

Why?

* Volume-to-node mapping must be global (not per namespace).

1. **CSIDriver**: Defines a CSI (Container Storage Interface) driver.

Why?

* Drivers are installed at cluster-level, not per namespace.

1. **CSINode**: Shows CSI drivers installed on a specific node.

Why?

* Node-level, cluster-wide object for storage management.

1. **PriorityClass**: Defines scheduling priority for Pods.

Why?

* Priority values must be comparable across all namespaces.

1. **MutatingWebhookConfiguration**: Admission webhooks that can mutate (change) objects before persistence.

Why?

* Hooks apply to the entire API server, not a namespace.

1. **ValidatingWebhookConfiguration**: Admission webhooks that validate objects before creation/update.

Why?

* Hooks apply to the entire API server, not a namespace.

1. **ClusterTrustBundle**: Holds CA bundles trusted by workloads.

Why?

* TLS trust roots apply cluster-wide.

1. **PodSecurityPolicy**: Defined security rules for pod creation.

Why?

* Security rules must be enforced globally.

Named scoped resources

* They exist inside a namespace (like Pods, Deployments, Services).

1. **Pod**: The smallest deployable unit in Kubernetes (runs containers).

Why?

* Pods belong to applications/environments → you don’t want Pods from different projects clashing → hence scoped to namespaces.

1. **ReplicaSet**:Ensures a certain number of Pod replicas are running.

Why?

* It manages Pods for an app inside a namespace.

1. **Deployment**: Manages ReplicaSets, provides rollout/rollback for Pods.

Why?

* Deployments are tied to application workloads → each project/team keeps its own in a namespace.

1. **StatefulSet**: Manages Pods that need stable identity & storage (e.g., databases).

Why?

* Databases or stateful apps are usually isolated by project → namespace-scoped.

1. **DaemonSet**: Ensures a Pod runs on every Node (or subset).

Why?

* Still belongs to a project/application, so scoped to a namespace.

1. **Job**:Runs Pods until a task finishes successfully.

Why?

* Jobs are workload-related and logically tied to a namespace.

1. **CronJob**: Runs Jobs on a schedule (like Linux cron).

Why?

* Each project/team schedules its own jobs, so scoped.

1. **Service**:Provides stable networking/DNS for a set of Pods.

Why?

* Services expose Pods that live in a namespace.

1. **ConfigMap**:Key-value pairs of configurations.

Why?

* Configs are per-application or per-environment → namespace-scoped.

1. **Secret**: Stores sensitive data (passwords, tokens).

Why?

* Security is critical → one team’s secrets shouldn’t be visible to others → namespace-scoped.

1. **ServiceAccount**: Provides an identity for Pods to interact with the API server.

Why?

* Service accounts are usually tied to workloads in a namespace.

1. **Role**: RBAC permissions inside a namespace.

Why?

* Many teams share a cluster → access control per namespace.

1. **RoleBinding**: Binds a Role to users/groups/service accounts in a namespac.

Why?

* Ensures access control is localized to a namespace.

1. **Ingress**: Exposes HTTP/S routes from outside to Services in a namespace.

Why?

* Ingress routes are tied to Services, which live inside namespaces.

1. **NetworkPolicy**: Defines how Pods in a namespace can communicate with each other or outside.

Why?

* Networking rules should apply at the namespace/application boundary.

1. **ResourceQuota**: Limits the total CPU, memory, and object count per namespace.

Why?

* Helps enforce fair usage between teams/projects.

1. **LimitRange**: Sets default/request/limit values for Pods/containers in a namespace.

Why?

* To enforce resource policies per team/project namespaces.