**Cluster scoped resources vs Namespace scoped resources**

**🔹 What are Kubernetes Resources?**

Definition:

* In Kubernetes, a resource is an API object that represents a piece of your cluster’s desired state or configuration.
* They are the “building blocks” that Kubernetes uses to manage your cluster.

When you run commands like kubectl get pods or kubectl apply -f deployment.yml, you are interacting with resources in the Kubernetes API.

**Cluster-scoped resources**

**What it is:**

* These are Kubernetes resources that exist at the **cluster level** and are not tied to any single namespace.
* They are global resources, visible across all namespaces.

**Why it is:**

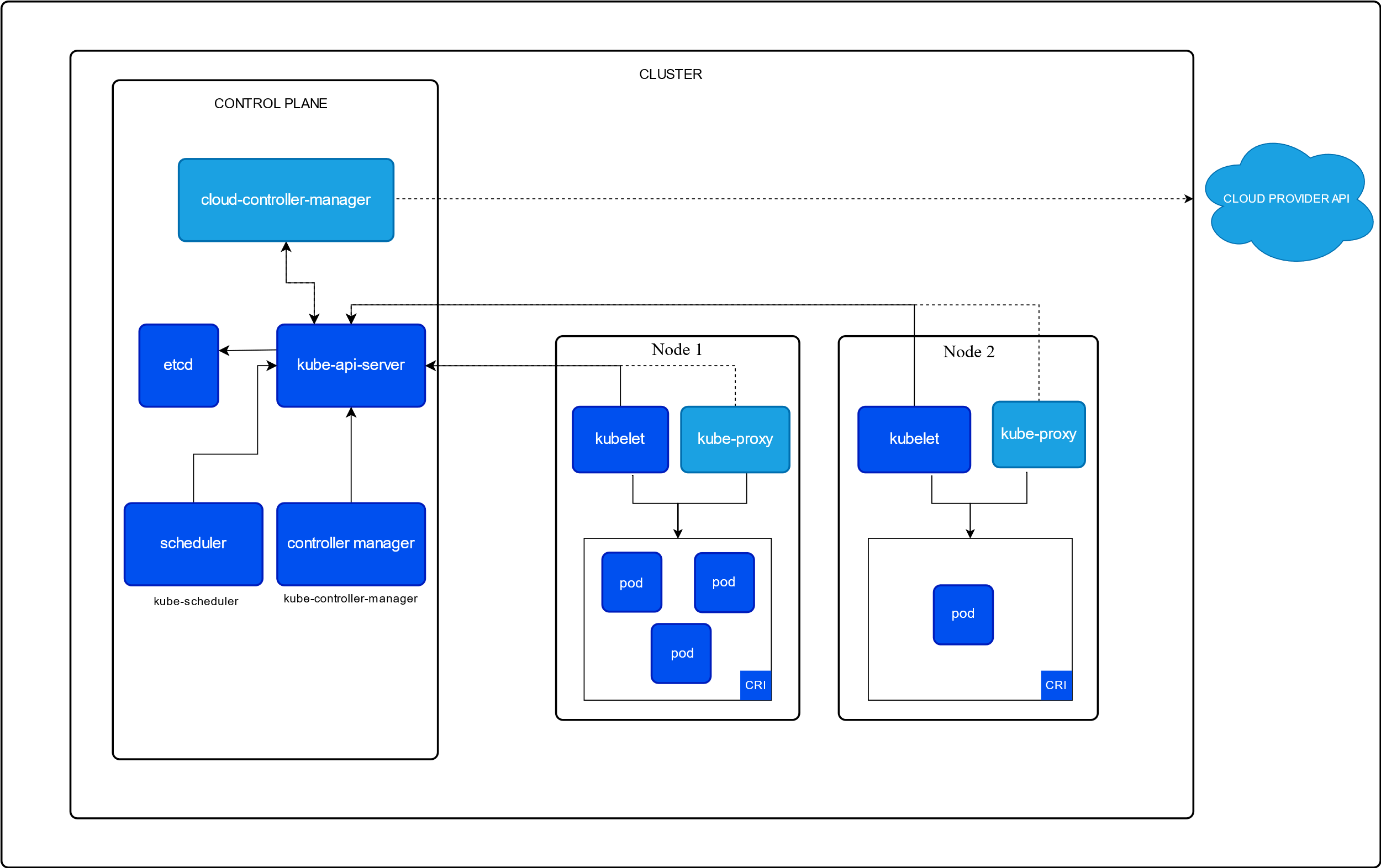
* Because these resources control things that must apply cluster-wide (like nodes, storage classes, RBAC policies, CRDs, etc.).
* They are designed this way to avoid duplication and to ensure consistent cluster-wide configurations.

**Characteristics:**

* Only one instance is needed for the whole cluster.
* They are accessible to all namespaces (directly or indirectly).
* They require cluster-level permissions to create/update (ClusterRoles).

**Examples & Explanations:**

* Nodes → Represent the worker machines in the cluster. Nodes can’t belong to a namespace, because workloads from multiple namespaces can be scheduled on the same node.
* StorageClass → Defines types of storage (SSD, HDD, cloud volumes). Storage must be consistent cluster-wide.
* PersistentVolume (PV) → Represents physical storage in the cluster. Any namespace can request it using a PVC.
* ClusterRole & ClusterRoleBinding → Provide RBAC permissions cluster-wide (e.g., an admin role that applies everywhere).
* CustomResourceDefinition (CRD) → Defines new resource types that can be used in all namespaces.



**Namespace-scoped resources**

**What it is:**

* These are resources that exist **within a namespace**.
* Each namespace provides an isolated “scope” for teams, apps, or projects.

**Why it is:**

* Because many workloads need logical isolation (for dev, test, prod, or per team).
* Namespace-scoped resources prevent naming conflicts and allow per-namespace policies (like quotas, RBAC).

**Characteristics:**

* You can create multiple copies of the same resource in different namespaces (e.g., Deployment named app1 in dev and prod).
* Easy to separate workloads per team or environment.
* Permissions can be applied per namespace using Role & RoleBinding.

**Examples & Explanations:**

* Pod → Runs application workloads. A pod in dev namespace is isolated from pods in prod.
* Service → Exposes a set of pods (only within its namespace unless explicitly configured).
* ConfigMap / Secret → Application-specific configs or credentials, tied to that namespace.
* Deployment / StatefulSet / DaemonSet → Workload controllers scoped per namespace.
* PersistentVolumeClaim (PVC) → Storage claim made by workloads in a namespace (bound to cluster-wide PV).
* Role & RoleBinding → Define permissions within a namespace only.

A screenshot of a computer

AI-generated content may be incorrect.

**Cluster vs Namespace Scoped Resources**

| **Scope** | **What it is** | **Why it is** | **Examples** |
| --- | --- | --- | --- |
| **Cluster-scoped** | Resources that apply to the **entire cluster** (not tied to any namespace). | Needed for **global configurations** and things that must be consistent across all namespaces. | - Node  - ClusterRole / ClusterRoleBinding  - StorageClass  - PersistentVolume (PV)  - CustomResourceDefinition (CRD) |
| **Namespace-scoped** | Resources that exist **inside a namespace** and are isolated to that scope. | Provides **separation for teams/apps**, prevents naming conflicts, allows per-namespace RBAC/quotas. | - Pod  - Deployment / StatefulSet / DaemonSet  - Service  - ConfigMap / Secret  - PersistentVolumeClaim (PVC)  - Role / RoleBinding |

**Why does Kubernetes separate resources this way?**

* **Separation of concerns:** Infrastructure admins manage cluster-scoped resources; app developers work with namespace-scoped ones.
* **Security & RBAC:** Prevents app developers from accidentally changing cluster-wide configurations.
* **Scalability:** Multiple teams can work in isolated namespaces without interfering with each other.
* **Reusability:** Cluster-wide resources (like storage classes) can be shared by multiple namespaces.

**Real-world analogy**

* Think of a **university campus (cluster)**.
  + **Cluster-scoped resources:** Campus-level rules, electricity, water, security (apply everywhere).
  + **Namespace-scoped resources:** Individual classrooms or departments (students, teachers, assignments) isolated within that department.

**Diagram Layout**

1. *Kubernetes Cluster*
   * This shows everything is inside one cluster.
2. **Inside the cluster**
   * **Namespace: prod**
     + Pods
     + Services
     + Deployments
     + ConfigMaps, PVCs
3. **Outside those namespace boxes** (but still inside cluster box), write:
   * **Cluster-scoped resources:**
     + Nodes
     + PersistentVolumes
     + StorageClasses
     + ClusterRoles

👉 The idea is:

* Namespace boxes contain **namespace-scoped resources**.
* Outside them, still inside the cluster, are **cluster-scoped resources** shared by all.