**🔐 What is Authorization in Kubernetes?**

As a reminder:

**Authorization** = "Now that I know who you are (authentication), **what are you allowed to do?**"

The **Kubernetes API server** uses different **authorization modes** (ways of checking permissions) to decide if someone can do something — like creating a pod or reading a secret.

There are **4 main types** of authorization used in Kubernetes:

**✅ 1. RBAC (Role-Based Access Control) — 📋 Most Common**

**🧠 What is it?**

RBAC is like a **permission system** based on:

* **Roles** (what actions are allowed)
* **Subjects** (who is allowed to do them)

You create **roles**, and then **assign them** to:

* users 👤
* groups 👥
* service accounts 🔐

**🧩 Example:**

1. ✅ You create a **Role** (e.g., pod-reader) that says:

rules:

- apiGroups: [""]

resources: ["pods"]

verbs: ["get", "list"]

1. ✅ Then you bind it to a user:

subjects:

- kind: User

name: alice

➡️ Now user alice can only **read** pods — not delete or edit.

**🛠️ When to Use RBAC:**

* ✅ Most clusters use RBAC by default.
* ✅ Best for **fine-grained control**.
* ✅ Works for both **users and service accounts**.

**👶 Simple Analogy:**

RBAC is like giving people **access cards**. You say:

* “This card allows access to the kitchen and lounge only.”
* Then hand it to Alice.

**✅ 2. ABAC (Attribute-Based Access Control) — 🧾 Legacy & Less Used**

**🧠 What is it?**

ABAC lets you define **rules in a file** based on **attributes** (like username, action, resource).

It’s **older** than RBAC and not dynamic — you have to edit a JSON file and restart the API server to make changes.

**🧩 Example Rule:**

{

"user": "bob",

"resource": "pods",

"verb": "delete",

"namespace": "dev"

}

➡️ This allows bob to delete pods only in the dev namespace.

**🛠️ When to Use ABAC:**

* Rarely used now.
* Only useful in **legacy systems** or where RBAC is not an option.

**⚠️ Limitations:**

* Harder to manage.
* No native CLI (kubectl) tools.
* Changes require API server restart.

**👶 Simple Analogy:**

ABAC is like writing rules on **paper** and pinning them to the wall. Every time you want to change something, you have to rewrite the paper and reboot the office.

**✅ 3. Webhook Authorization — ⚙️ Fully Custom**

**🧠 What is it?**

This lets you **plug in your own authorization server** using HTTP.

Kubernetes sends a request to **your external service**, and your service replies: ✅ "yes" or ❌ "no".

**🧩 Example Use Case:**

You build a **custom policy**:

* Allow deleting pods only **after 6 PM**
* Allow access only if the user’s team lead approved the request
* Allow access based on a company HR system

You can write this logic in Python, Go, Node.js — and plug it into Kubernetes as a **Webhook Authorizer**.

**🛠️ When to Use Webhooks:**

* You need **custom policies** beyond what RBAC can do.
* You integrate with **external systems** (HR, approval workflows, etc.)

**👶 Simple Analogy:**

Webhook Authorization is like calling your manager before doing anything:

* "Hey, can I delete this pod?"
* Your manager replies "Yes" or "No"
* Then you proceed.

**✅ 4. Node Authorization — 🧱 For Internal Node Security**

**🧠 What is it?**

This mode is only for **Kubernetes nodes themselves** (not users).

It controls what each **kubelet (the agent running on nodes)** is allowed to do, such as:

* Read its own Pod specs
* Access only secrets mounted to its pods
* Watch pods scheduled to itself

This ensures that **Node A can’t access Pod data from Node B**.

**🛠️ When is it Used?**

* Always enabled by default in secure clusters.
* Keeps each node’s access **strictly limited**.

**👶 Simple Analogy:**

Node Authorization is like giving each janitor a key to **only their assigned floor** in a building. They can't open rooms on other floors.

**🧾 Summary Table: All 4 Authorization Types**

| **Type** | **What it does** | **Who it applies to** | **Strength** | **Common Use** |
| --- | --- | --- | --- | --- |
| **RBAC** | Role & permission-based | Users, service accounts | ⭐⭐⭐⭐⭐ | ✅ Modern clusters |
| **ABAC** | Rule-based on attributes (JSON file) | Users | ⭐⭐ | ❌ Legacy use only |
| **Webhook** | Sends request to external system | Users, groups, SA | ⭐⭐⭐⭐ | ✅ Custom rules |
| **Node** | Secures kubelets & node access | Kubernetes nodes | ⭐⭐⭐⭐ | ✅ Always used |

**✅ Final Recap for Interns:**

* **RBAC** = "You have a keycard with specific room access."
* **ABAC** = "Access rules are written down and hard to change."
* **Webhook** = "You call someone to ask for permission each time."
* **Node** = "Each cleaner only has access to their assigned floor."

A **ClusterRole** in Kubernetes is like a regular Role, but it applies **across the entire cluster**, not just within a specific namespace. It defines a set of permissions (like creating pods, accessing secrets, or modifying nodes) that can be granted to users, groups, or service accounts **regardless of namespace**. ClusterRoles are commonly used for granting access to **non-namespaced resources** (e.g., nodes, persistent volumes, or cluster-wide settings) or for giving **global permissions** that apply to all namespaces. You bind a ClusterRole using a **ClusterRoleBinding** to assign those permissions at the cluster level.