**LAB#4-Objective:**

**To implement fine-grained access control in a Kubernetes cluster using ClusterRoles and ClusterRoleBindings.**

**Tasks:**

**1-create clusterrole, bind this cluster role to user and integrate with service account ,**

**2- Test the access connectivity:**

**Prerequisites:**

**Install kubectl**

**Install eks**

**Install awscli**

**Add access key**

**Add roles for cluster and node permissions**

Let's create a ClusterRole named pod-reader that allows read-only access to pods

| apiVersion: rbac.authorization.k8s.io/v1 kind: ClusterRole metadata:  name: pod-reader rules: - apiGroups: [""]  resources: ["pods"]  verbs: ["get", "list","create",delete] |
| --- |

Next, bind the pod-reader ClusterRole to a specific user. For demonstration purposes, let's bind it to a user named john:

| apiVersion: rbac.authorization.k8s.io/v1 kind: ClusterRoleBinding metadata:  name: read-pods subjects: - kind: User  name: john  apiGroup: rbac.authorization.k8s.io roleRef:  kind: ClusterRole  name: pod-reader  apiGroup: rbac.authorization.k8s.io |
| --- |

Create a ServiceAccount named pod-reader-sa and associate it with the pod-reader ClusterRole:

| apiVersion: v1 kind: ServiceAccount metadata:  name: pod-reader-sa --- apiVersion: rbac.authorization.k8s.io/v1 kind: ClusterRoleBinding metadata:  name: read-pods-sa subjects: - kind: ServiceAccount  name: pod-reader-sa  namespace: default roleRef:  kind: ClusterRole  name: pod-reader  apiGroup: rbac.authorization.k8s.io |
| --- |

### Test the Access

| kubectl auth can-i create pods --all-namespaces |
| --- |

* **Documentation : Understanding ClusterRoles:** Definition: ClusterRoles are a set of permissions that define access levels across the entire Kubernetes cluster.
  + Scope: They apply globally and are not namespace-specific, enabling control over cluster-wide resources.
* Defining ClusterRoleBindings:
  + Association of Roles: ClusterRoleBindings associate ClusterRoles to specific users, groups, or service accounts within the cluster.
  + Binding to Subjects: They bind ClusterRoles to subjects, defining who has access to which resources within the cluster.
* Creating Custom ClusterRoles:
  + Customization: Creating custom ClusterRoles allows for fine-tuning permissions, tailoring access based on specific requirements.
  + Granular Permissions: Custom roles can grant or restrict access to specific resources, API groups, or verbs.
* Applying ClusterRoleBindings:
  + Precise Authorization: Applying ClusterRoleBindings to specific entities or service accounts ensures that only authorized entities have access to defined resources or actions.

**Conclusion of the lab:**

**The implementation of fine-grained access control using ClusterRoles and ClusterRoleBindings in Kubernetes establishes a robust authorization mechanism, enhancing security and control within the cluster.**

* **Granular Access Management: Utilizing ClusterRoles enables defining granular permissions, specifying access levels for resources, API groups, and actions across the entire cluster.**
* **Controlled Authorization: Binding ClusterRoles through ClusterRoleBindings to specific users, groups, or service accounts ensures precise authorization, limiting access to resources based on defined roles.**
* **Customization for Specific Needs: The ability to create custom ClusterRoles allows for tailoring permissions to meet specific application or organizational requirements, promoting least privilege access.**

**Implementing these access control mechanisms facilitates the establishment of a secure and controlled environment within the Kubernetes cluster. It ensures that only authorized entities have access to specific resources, minimizing the risk of unauthorized actions or breaches.**

**Continued monitoring, periodic reviews of access controls, and fine-tuning of ClusterRoles and ClusterRoleBindings further bolster the security posture of the Kubernetes cluster, ensuring compliance with organizational policies and best practices.**