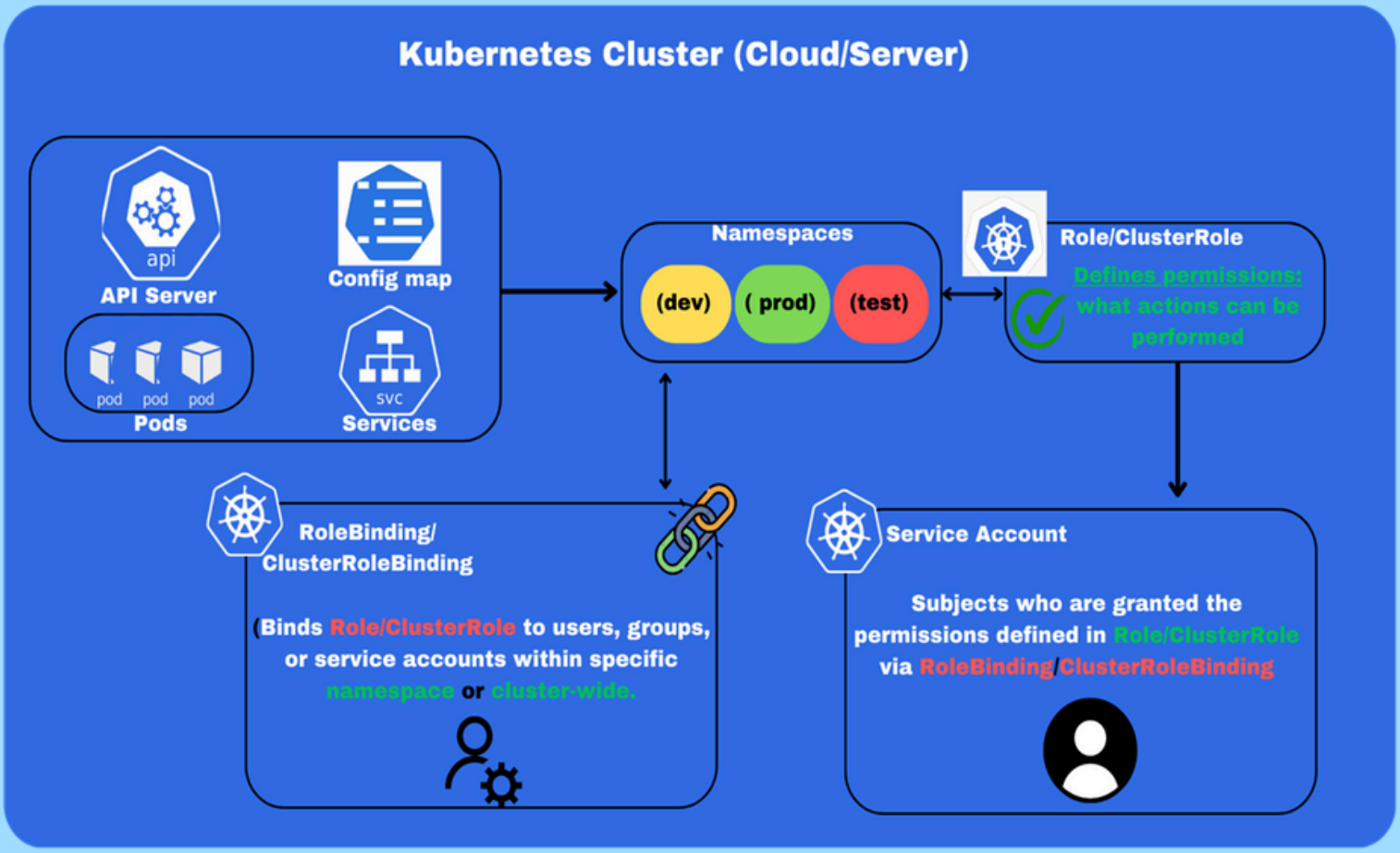
DevOps Shack

Role-Based Access Control in Kubernetes

How Kubernetes Controls Who Can Do What

Introduction to RBAC

Role-Based Access Control (RBAC) is a method used by Kubernetes to regulate access to resources within a cluster. With RBAC, permissions can be assigned to users, groups, or service accounts to ensure that they only have the access they need to perform their tasks, improving security and reducing the risk of unauthorized operations.



Kubernetes RBAC helps manage who can **read, modify, or administer** the different parts of the cluster. By assigning specific roles to users or services, you can ensure that they follow the principle of **least privilege**, where access is granted only for the minimum tasks needed.

Key Components of Kubernetes RBAC

1. **Role/ClusterRole**:
    - A **Role** defines a set of permissions (verbs like get, list, create, etc.) within a specific namespace.
    - A **ClusterRole** is similar but applies cluster-wide or across all namespaces.
2. **RoleBinding/ClusterRoleBinding**:
    - **RoleBinding** associates a Role with users, groups, or service accounts in a particular namespace.
    - **ClusterRoleBinding** associates a ClusterRole with users, groups, or service accounts across the entire cluster.
3. **Users, Groups, and Service Accounts**:
    - Subjects in RBAC, like users, groups, or service accounts, are assigned roles and bindings, which control their access levels.

How RBAC Works in Kubernetes

RBAC operates by associating users with roles via **RoleBinding** or **ClusterRoleBinding**. A **Role** or **ClusterRole** contains rules that define which actions (verbs) can be performed on which resources (objects).

For example:

- A **Role** may allow a user to only view Pods within a certain namespace.
- A **ClusterRole** could give an admin full access to all resources across the cluster.

Example Structure:

- **Role** or **ClusterRole**: Defines **what** actions can be performed on **which** resources.
- **RoleBinding** or **ClusterRoleBinding**: Specifies **who** can access resources defined by the roles.

Creating a Role in Kubernetes

A **Role** defines access within a specific namespace. Below is an example YAML manifest for a **Role** that grants read-only access to Pods within a namespace:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
 namespace: dev
 name: pod-reader
rules:
- apiGroups: [""] # "" indicates the core API group
 resources: ["pods"]
 verbs: ["get", "list", "watch"]
```

In this example:

- The Role pod-reader allows the verbs (get, list, watch) on pods within the dev namespace.

Creating a RoleBinding

A **RoleBinding** binds a Role to a specific user, group, or service account in a particular namespace. Here's a YAML manifest that binds the pod-reader role to a user:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
 name: read-pods
 namespace: dev
subjects:
- kind: User
 name: "jane" # Replace with the actual username
 apiGroup: rbac.authorization.k8s.io
roleRef:
 kind: Role # Can be "Role" or "ClusterRole"
 name: pod-reader # Role being assigned
 apiGroup: rbac.authorization.k8s.io
```

This **RoleBinding**:

- Binds the pod-reader role to the user jane within the dev namespace.

Creating a ClusterRole

A **ClusterRole** is used when access across the entire cluster or multiple namespaces is required. Here's an example of a ClusterRole that grants admin access to all resources:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
 name: cluster-admin
rules:
```

```
- apiGroups: [""]
resources: ["*"]
 verbs: ["*"] # Allows all actions (create, delete, update, etc.)
```

In this example:

- The ClusterRole cluster-admin grants full access (* for all verbs) to all resources in all API groups.

Creating a ClusterRoleBinding

To assign a **ClusterRole** to a user or group across the entire cluster, a **ClusterRoleBinding** is used. Here's an example:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
 name: admin-access
subjects:
- kind: User
 name: "admin" # Replace with the actual username
 apiGroup: rbac.authorization.k8s.io
roleRef:
 kind: ClusterRole
 name: cluster-admin # ClusterRole being assigned
 apiGroup: rbac.authorization.k8s.io
```

This **ClusterRoleBinding**:

- Grants the cluster-admin ClusterRole to the user admin, giving them cluster-wide administrative privileges.

Assigning Permissions to a Service Account

A **service account** is often used to assign roles to applications running in the cluster. Here's how you assign a role to a service account:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
 name: read-pods-to-service-account
 namespace: dev
subjects:
- kind: ServiceAccount
 name: my-service-account
 namespace: dev
roleRef:
 kind: Role
 name: pod-reader
 apiGroup: rbac.authorization.k8s.io
```

In this case:

- The service account my-service-account is bound to the pod-reader role, allowing it to read Pods in the dev namespace.

Listing Roles and RoleBindings

To view the roles and bindings in your cluster, you can use the following kubectl commands:

- List all Roles in a namespace:

`kubectl get roles -n <namespace>`

- List all RoleBindings in a namespace:

`kubectl get rolebindings -n <namespace>`

- List all ClusterRoles:

`kubectl get clusterroles`

- List all ClusterRoleBindings:

`kubectl get clusterrolebindings`

Best Practices for RBAC in Kubernetes

1. **Principle of Least Privilege**: Always assign the minimum level of access needed. Avoid assigning broad permissions to users or service accounts.
2. **Namespace Isolation**: Assign Roles within specific namespaces to limit the scope of access, unless absolutely necessary for it to be cluster-wide.
3. **Audit Role and RoleBinding Usage**: Regularly review roles, bindings, and their permissions to ensure they are aligned with current security policies.
4. **Use Service Accounts for Applications**: When assigning permissions to applications, use service accounts rather than user credentials to ensure scoped access for each application.

Conclusion

Kubernetes RBAC is a powerful mechanism to control access to resources within a cluster, enabling you to maintain security and manage permissions efficiently. By defining Roles and ClusterRoles and binding them to users, groups, or service accounts, you can ensure that each entity in your cluster has only the necessary permissions. Following best practices such as applying the principle of least privilege and namespace isolation will enhance your Kubernetes security posture significantly.