Understanding Kubernetes Auth

Relevant Documentation

- Authenticating
- Controlling Access to the Kubernetes API
- Using RBAC Authorization

Exam Tips

- Normal users usually authenticate using client certificates, while ServiceAccounts usually use tokens.
- Authorization for both normal users and ServiceAccounts can be managed using role-based access control (RBAC).
- Roles and ClusterRoles define a specific set of permissions.
- RoleBindings and ClusterRoleBindings tie Roles or ClusterRoles to users/ServiceAccounts.

Lesson Reference

Log in to the **control plane node**.

Create a Role that provides permission to list Pods.

```
vi list-pods-role.yml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: list-pods-role
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["list"]
```

```
kubectl apply -f list-pods-role.yml
```

Create a RoleBinding to bind the new Role to the `my-sa` ServiceAccount. **Note:** This ServiceAccount was created in a previous lesson.

```
vi list-pods-rb.yml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: list-pods-rb
subjects:
- kind: ServiceAccount
  name: my-sa
  namespace: default
roleRef:
  kind: Role
  name: list-pods-role
  apiGroup: rbac.authorization.k8s.io
```

```
kubectl apply -f list-pods-rb.yml
```

Check the logs for `sa-pod`.

```
kubectl logs sa-pod
```

Now that permissions have been provided to the ServiceAccount using RBAC, the log should indicate that the Pod is able to successfully access the API and retrieve a list of Pods in the `default` Namespace.