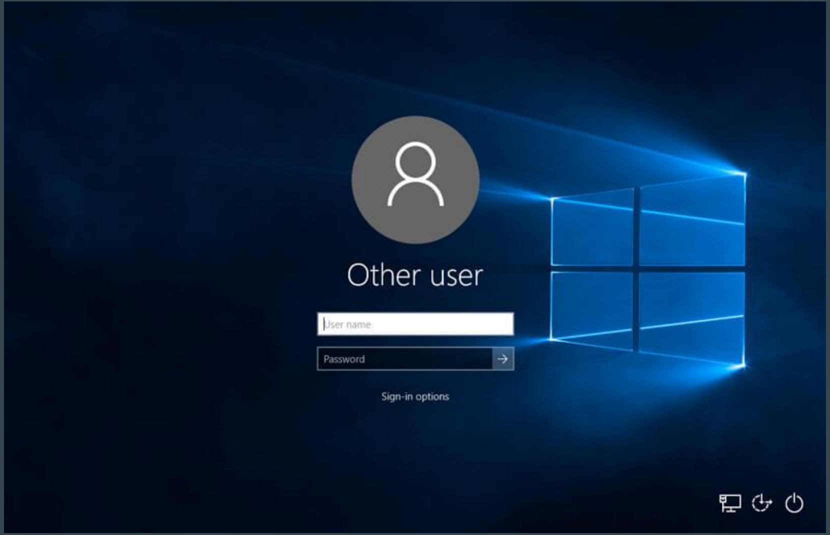
Authentication in Kubernetes

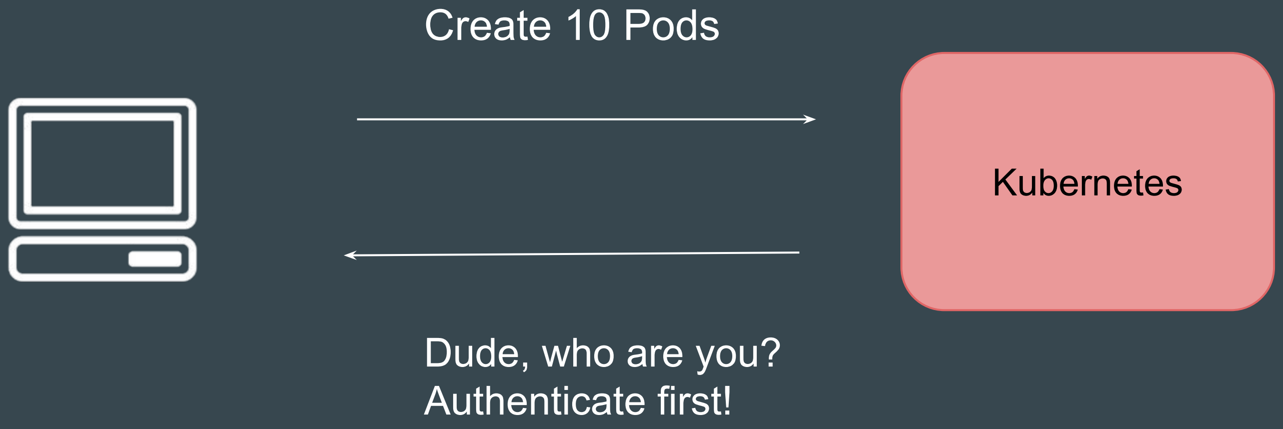
Basics of Authentication

Authentication is the process of verifying a user's identity before granting them access to a system or resource



Accessing Resources in Kubernetes

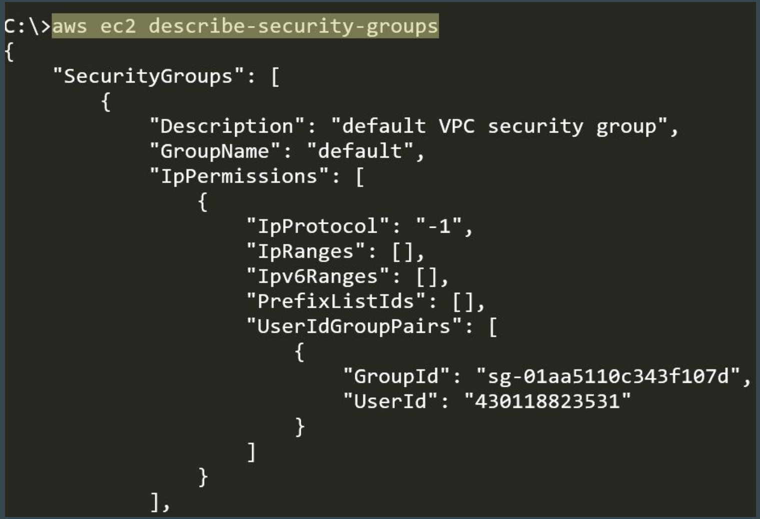
To access resources in Kubernetes cluster, we have to authenticate first.



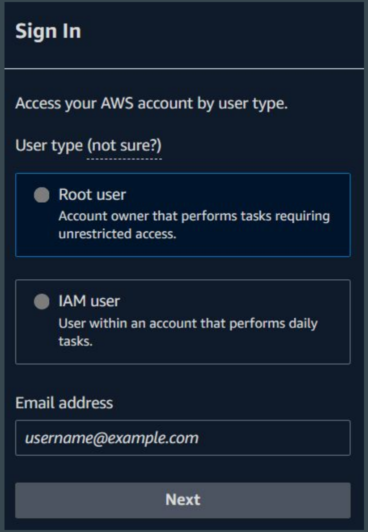
Analogy of AWS

In AWS, you can authenticate using multiple set of methods.

1. Username and Passwords.
2. Access Key and Secret Keys



```
C:\>aws ec2 describe-security-groups
{
    "SecurityGroups": [
        {
            "Description": "default VPC security group",
            "GroupName": "default",
            "IpPermissions": [
                {
                    "IpProtocol": "-1",
                    "IpRanges": [],
                    "Ipv6Ranges": [],
                    "PrefixListIds": [],
                    "UserIdGroupPairs": [
                        {
                            "GroupId": "sg-01aa5110c343f107d",
                            "UserId": "430118823531"
                        }
                    ]
                }
            ],
```



Sign In

Access your AWS account by user type.

User type (not sure?)

● Root user
Account owner that performs tasks requiring unrestricted access.

● IAM user
User within an account that performs daily tasks.

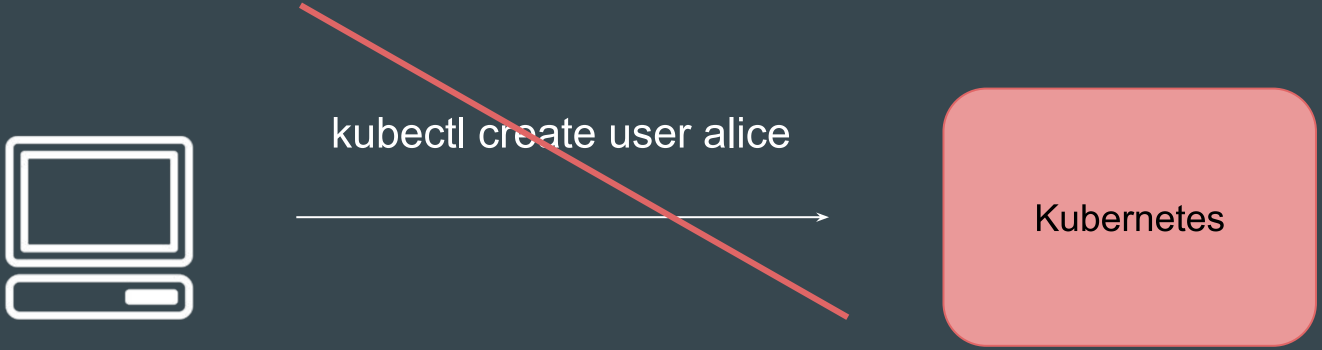
Email address

username@example.com

Point to Note - Kubernetes

Kubernetes does not manage the user accounts natively.

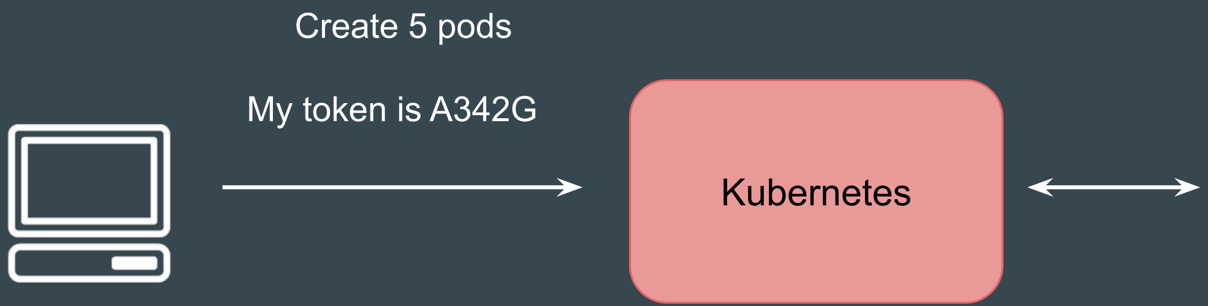
Normal users cannot be added to a cluster through an API call



Authentication in Kubernetes

Kubernetes supports several authentication methods such as:

Client Certificates, Static Token Authentication, Service Account Tokens etc

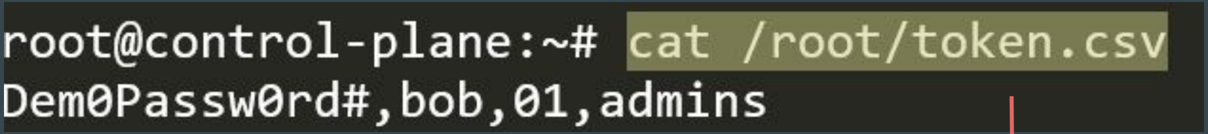


token	user
A342G	alice
BPRQ	bob

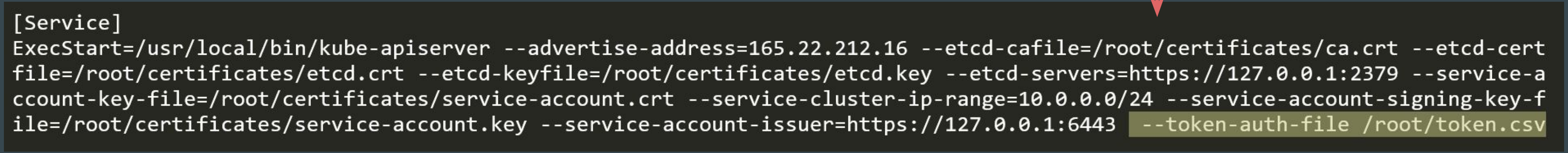
Example 1 - Static Token File

The API server reads bearer tokens from a file provided.

The token file is a csv file with a minimum of 3 columns: token, user name, user uid



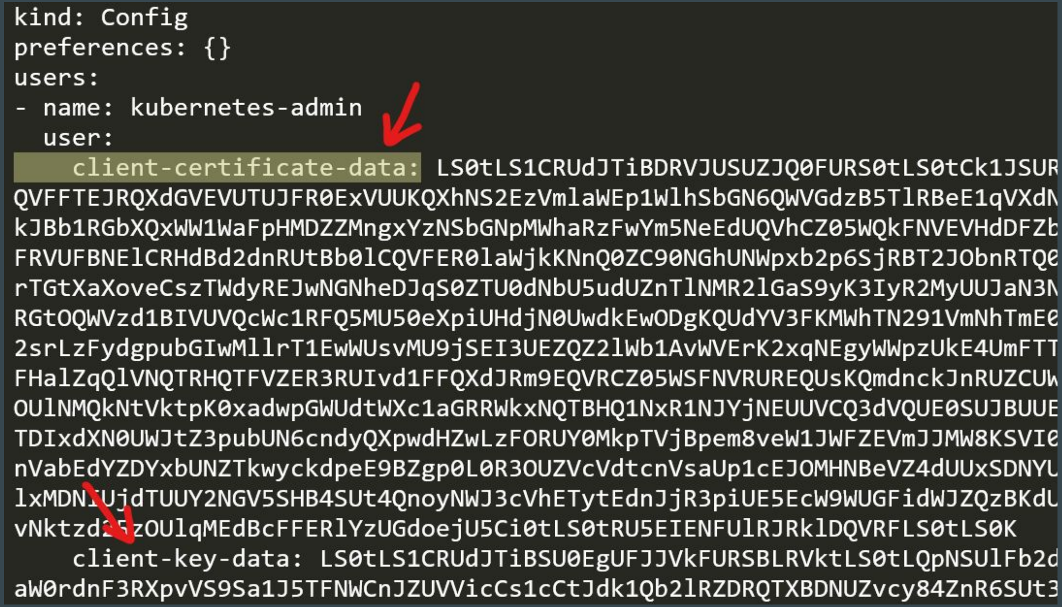
```
root@control-plane:~# cat /root/token.csv
Dem0Passw0rd#,bob,01,admins
```



```
[Service]
ExecStart=/usr/local/bin/kube-apiserver --advertise-address=165.22.212.16 --etcd-cafile=/root/certificates/ca.crt --etcd-cert
file=/root/certificates/etcd.crt --etcd-keyfile=/root/certificates/etcd.key --etcd-servers=https://127.0.0.1:2379 --service-a
ccount-key-file=/root/certificates/service-account.crt --service-cluster-ip-range=10.0.0.0/24 --service-account-signing-key-f
ile=/root/certificates/service-account.key --service-account-issuer=https://127.0.0.1:6443  --token-auth-file /root/token.csv
```

Example 2 - X509 Certificates

Uses the client certificates for authentication.



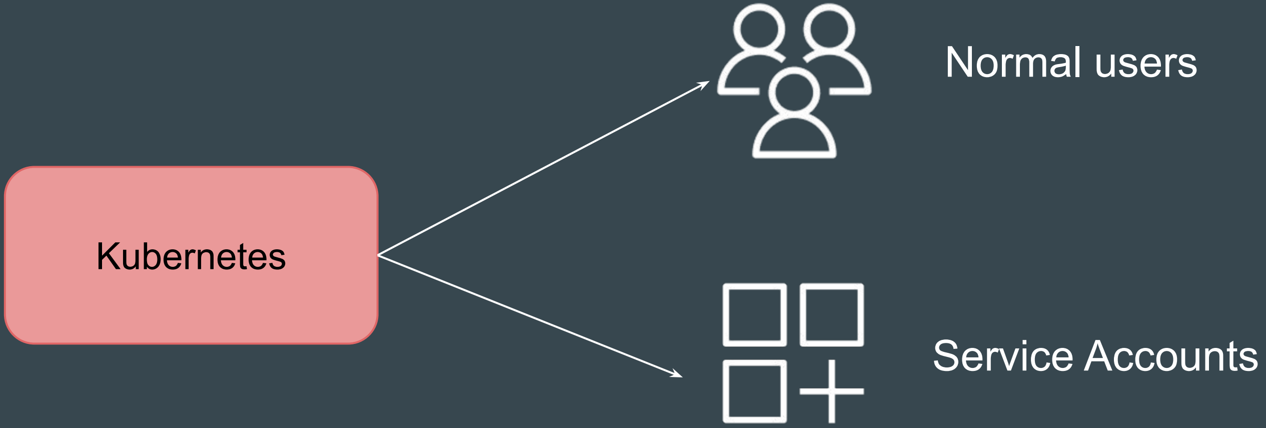
```
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUR
QVFFTEJRQXdGVEVUTUJFR0ExVUUKQXhNS2EzVmlaWEp1WlhSbGN6QWVGdzB5TlRBeE1qVXdN
kJBb1RGbXQxWW1WaFpHMDZZMngxYzNSbGNpMWhaRzFwYm5NeEdUQVhCZ05WQkFNVEVHdDFZb
FRVUFBNElCRHdBd2dnRUtBb0lCQVFER0laWjkKNnQ0ZC90NGhUNWpxb2p6SjRBT2JObnRTQ0
rTGtXaXoveCszTWdyREJwNGNheDJqS0ZTU0dNbU5udUZnTlNMR2lGaS9yK3IyR2MyUUJaN3N
RGtOQWVzd1BIVUVQcWc1RFQ5MU50eXpiUHdjN0UwdkEwODgKQUdYV3FKMWhTN291VmNhTmE0
2srLzFydgpubGIwMllrT1EwWUsvMU9jSEI3UEZQZ2lWb1AvWVErK2xqNEgyWWpzUkE4UmFTT
FHalZqQlVNQTRHQTFVZER3RUIvd1FFQXdJRm9EQVRCZ05WSFNVRUREQUsKQmdnckJnRUZCUW
OUlNMQkNtVktpK0xadwpGWUdtWXc1aGRRWkxNQTBHQ1NxR1NJYjNEUUVCQ3dVQUE0SUJBUUE
TDIxdXN0UWJtZ3pubUN6cndyQXpwdHZwLzFORUY0MkpTVjBpem8veW1JWFZEVmJJMW8KSVI0
nVabEdYZDYxbUNZTkwyckdpeE9BZgp0L0R3OUZVcVdtcnVsaUp1cEJOMHNBeVZ4dUUxSDNYU
lxMDNIUjdTUUY2NGV5SHB4SUt4QnoyNWJ3cVhETytEdnJjR3piUE5EcW9WUGFidWJZQzBKdU
vNktzd25zOUlqMEdBcFFERlYzUGdoejU5Ci0tLS0tRU5EIENFUlRJRklDQVRFLS0tLS0K
    client-key-data: LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFb2d
aW0rdnF3RXpvVS9Sa1J5TFNWCnJZUVVicCs1cCtJdk1Qb2lRZDRQTXBDNUZvcy84ZnR6SUt3
```

Categories of Users

Kubernetes Clusters have two categories of users:

1. Normal Users (for humans)

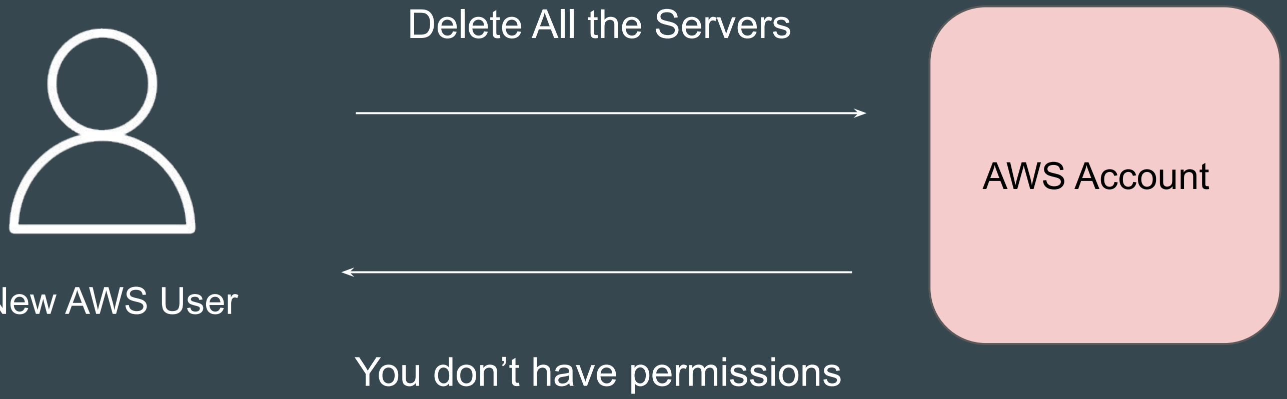
2. Service Accounts (for apps)



Authorization

Basics of Authorization

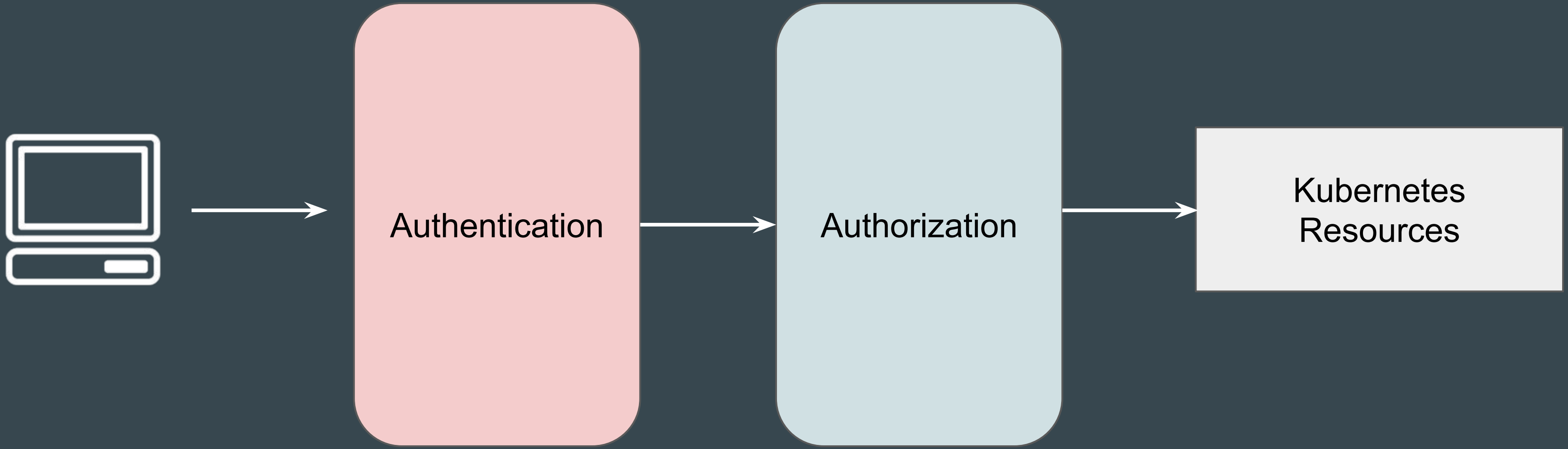
Authorization is the process of determining what an authenticated user or entity is allowed to do



Authorization in Kubernetes

Kubernetes authorization takes place following authentication.

Usually, a client making a request must be authenticated (logged in) before its request can be allowed.



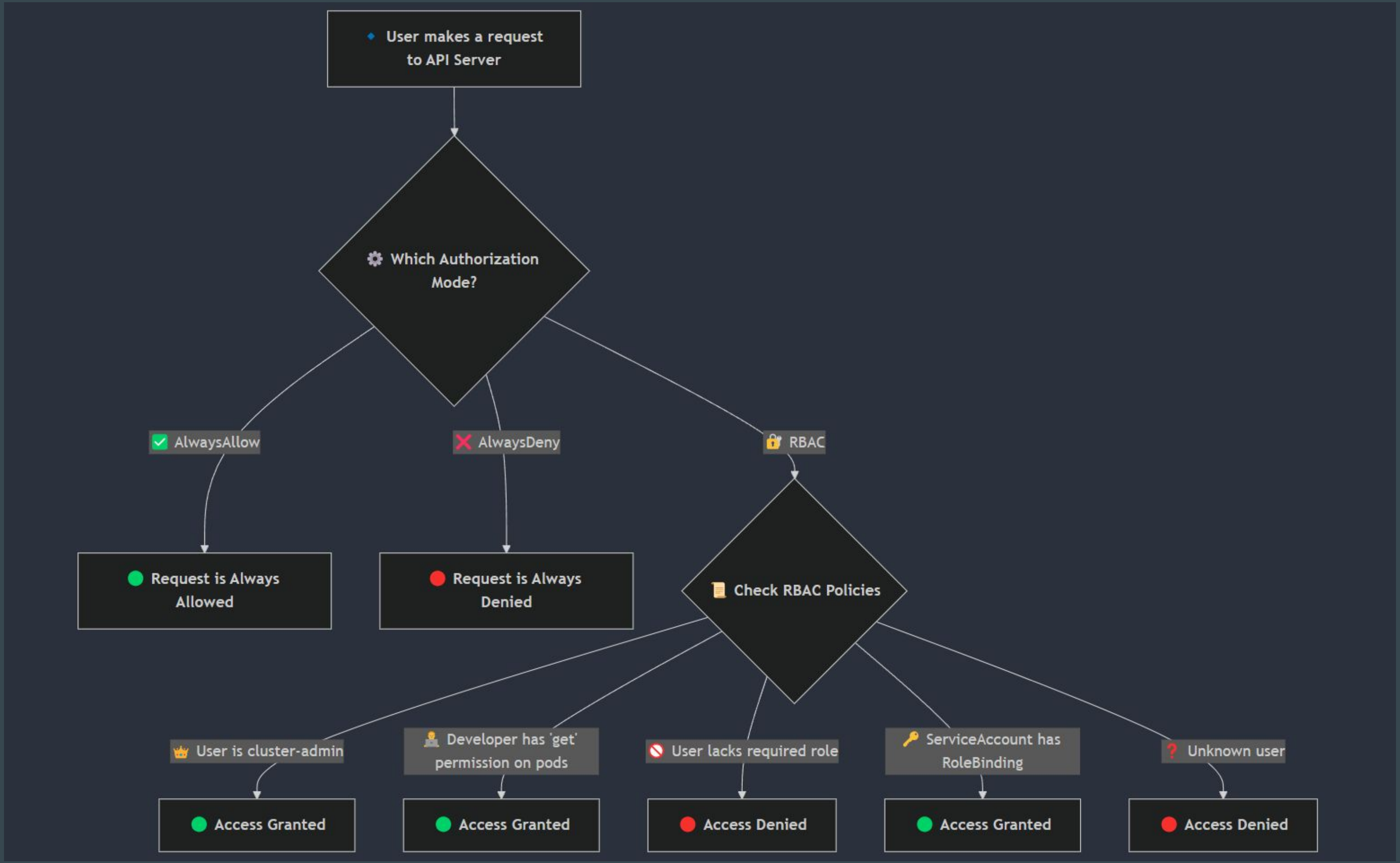
Authorization Modes

The Kubernetes API server may authorize a request using one of several authorization modes. Some of these include:

Authorization Mode	Description
AlwaysAllow	This mode allows all requests, which brings security risks. Use this authorization mode only for testing.
AlwaysDeny	This mode blocks all requests. Use this authorization mode only for testing.
RBAC	Defines set of permissions based on which access is granted. Recommended for Production.

Point to Note

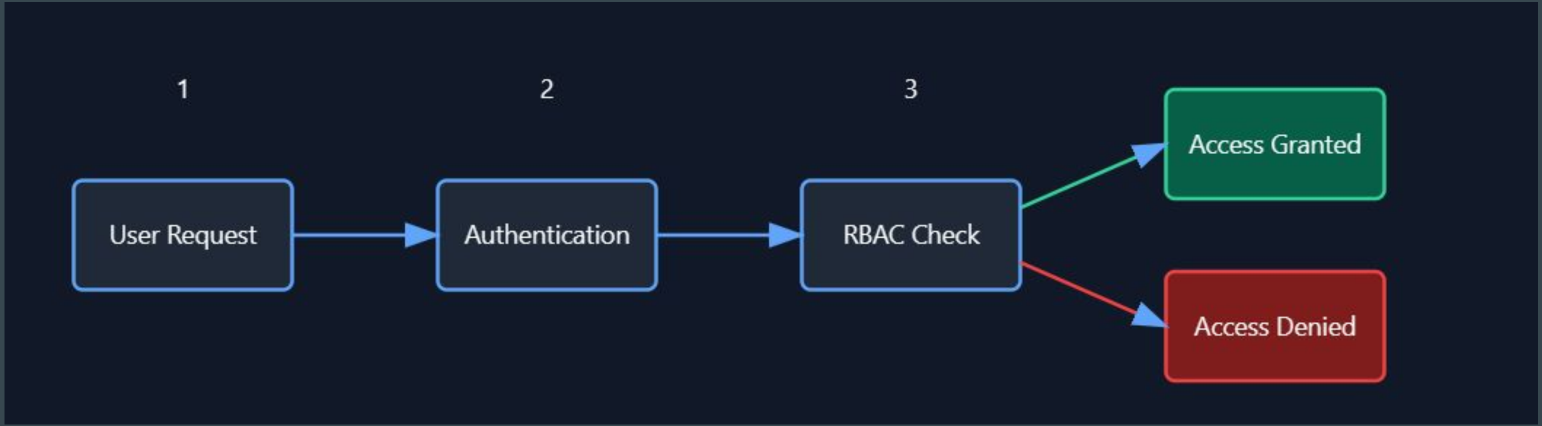
In Kubernetes, if the authorization mode is not explicitly defined in the API server configuration, the default mode used is AlwaysAllow.


🔹 User makes a request to API Server
⚙️ Which Authorization Mode?
✅ AlwaysAllow
❌ AlwaysDeny
🔐 RBAC
🟢 Request is Always Allowed
🔴 Request is Always Denied
📜 Check RBAC Policies
👑 User is cluster-admin
🧑‍💻 Developer has 'get' permission on pods
🚫 User lacks required role
🔑 ServiceAccount has RoleBinding
❓ Unknown user
🟢 Access Granted
🟢 Access Granted
🔴 Access Denied
🟢 Access Granted
🔴 Access Denied

Role-Based Access Control (RBAC)

Setting the Base

RBAC allows us to control what actions users and service accounts can perform on resources within your cluster.



Basic Workflow

In the below diagram, we have a list of users in Table 1 and list of permissions in Table 2.

We have to bind these together for users to get the defined permissions.

Users
Alice
Bob

Bind ⟷

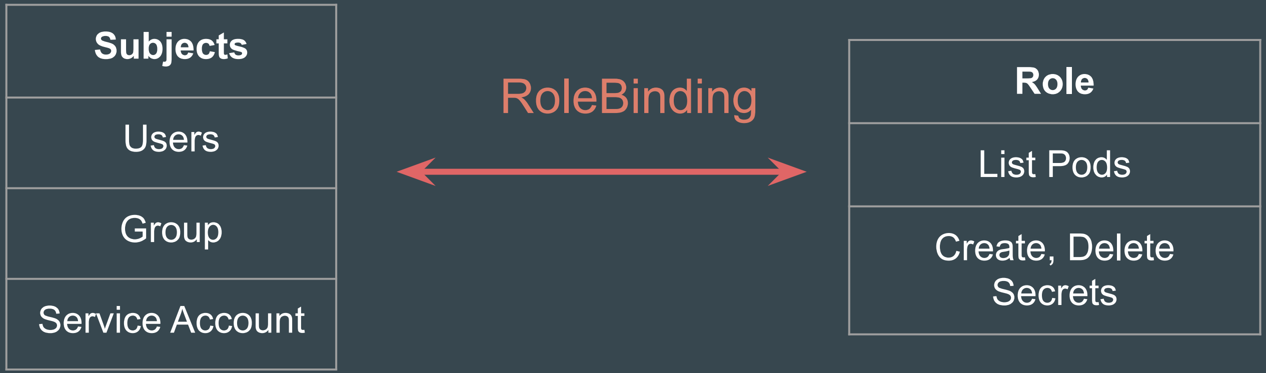
Permissions
List Pods
Create, Delete Secrets

3 Important Concept

Role defines a set of permissions.

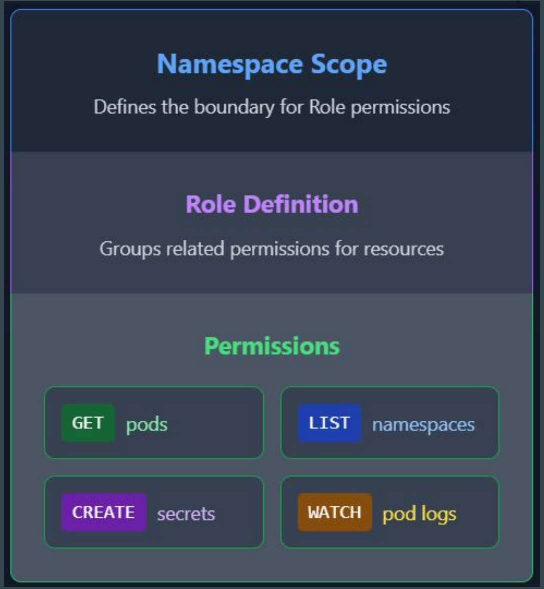
Subjects can be user, groups, service account.

RoleBinding ties the permission defined in the role to subjects like Users.



Introducing Roles

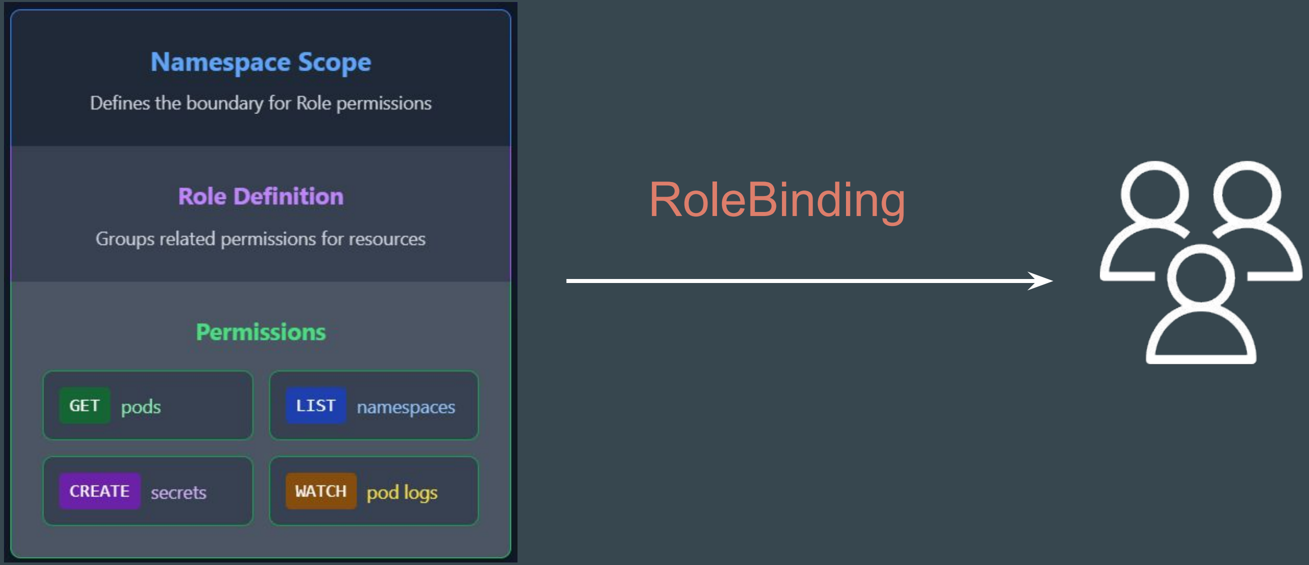
A Role always sets permissions within a particular namespace.



Introducing RoleBinding

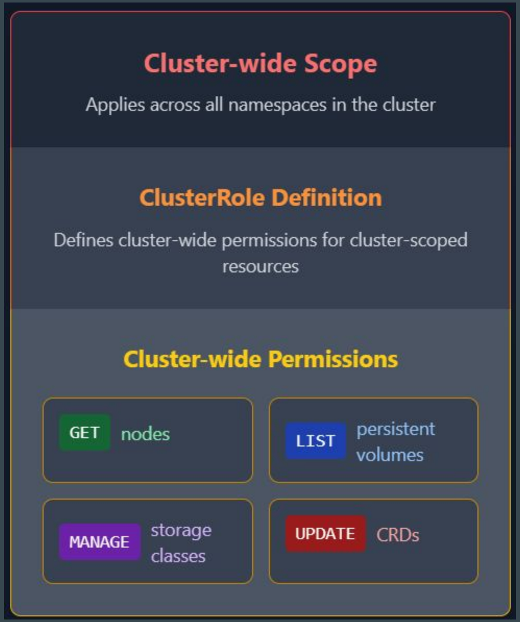
RoleBinding associates a Role with a user, group, or service account within a specific namespace.

It grants the defined permissions to the subjects in that namespace.



ClusterRole and ClusterRoleBinding

Similar to Role and RoleBinding, but the main difference is that the permissions granted by a ClusterRole apply across all namespaces in the cluster. ClusterRoleBinding connects ClusterRole to Subjects.



Cluster-wide Scope

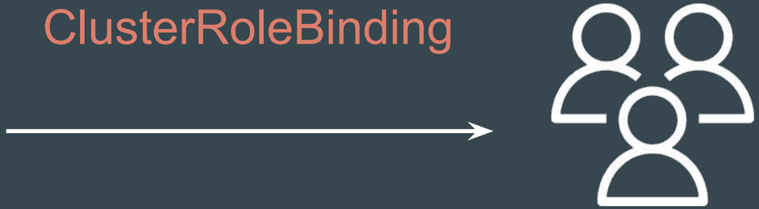
Applies across all namespaces in the cluster

ClusterRole Definition

Defines cluster-wide permissions for cluster-scoped resources

Cluster-wide Permissions

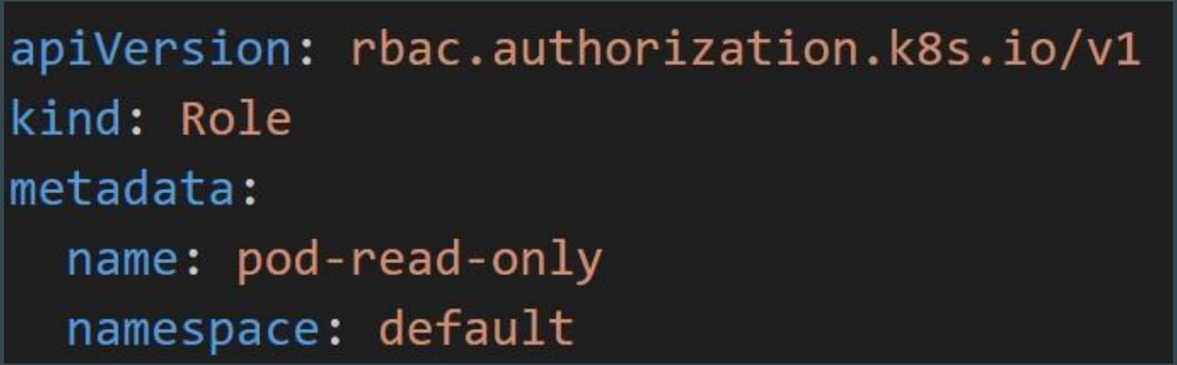
GET nodes	LIST persistent volumes
MANAGE storage classes	UPDATE CRDs



Practical - Role and RoleBinding

Basic Structure of Role Manifest

The following image represents the basic structure of the first part of a Role manifest file.

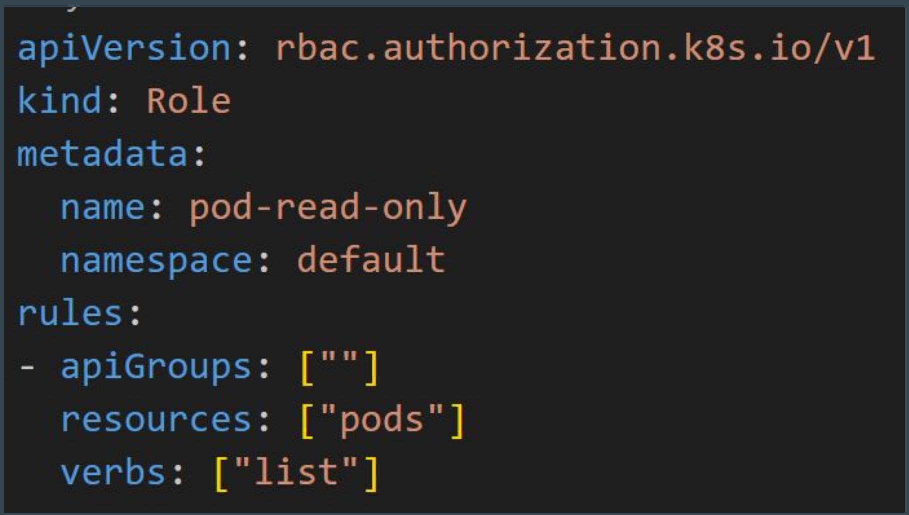


```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-read-only
  namespace: default
```

Defining Rules in Role Manifest

The rules field is a list of policies that define the permissions granted by the Role.

Each rule specifies which actions (verbs) are allowed on which resources (API objects).



```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-read-only
  namespace: default
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["list"]
```

1 - API Groups

apiGroups specify which API group the rule applies to.

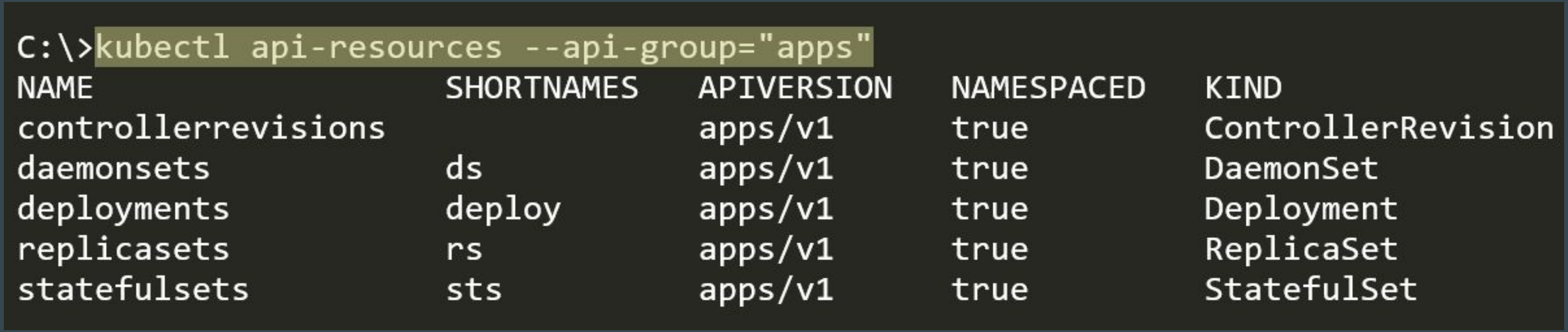
Kubernetes APIs are categorized into different API groups.

API Groups	**Description**
"" (empty string)	Refers to the core API group (e.g., pods, services, configmaps etc).
apps	Refers to the apps API group (e.g., deployments, daemonsets,replicasets)
batch	Includes Jobs, CronJobs.
networking.k8s.io	Handles Ingress and Network Policies.

2 - Resources

This field specifies which Kubernetes resources the rule applies to.

These resources belong to the specified API group.



```
C:\>kubectl api-resources --api-group="apps"
NAME                  SHORTNAMES   APIVERSION   NAMESPACED   KIND
controllerrevisions                apps/v1      true         ControllerRevision
daemonsets            ds           apps/v1      true         DaemonSet
deployments           deploy       apps/v1      true         Deployment
replicasets           rs           apps/v1      true         ReplicaSet
statefulsets          sts          apps/v1      true         StatefulSet
```

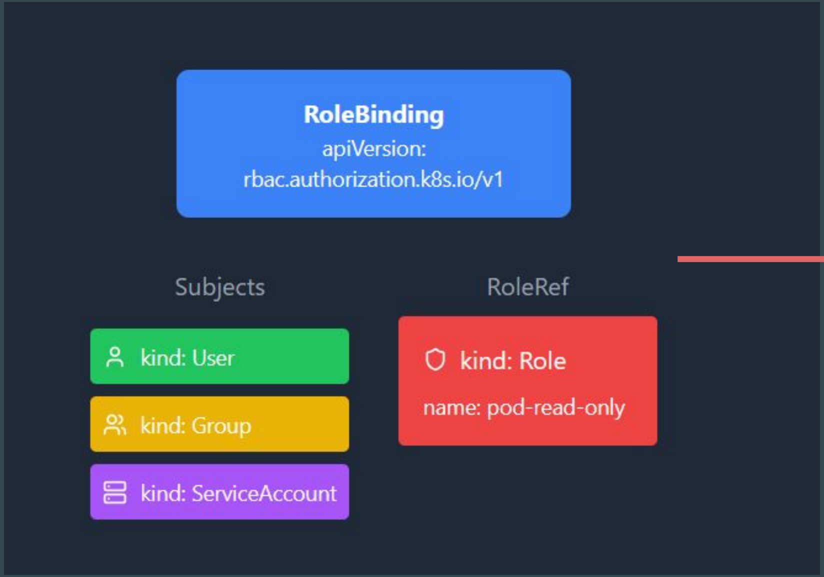
3 - Verbs

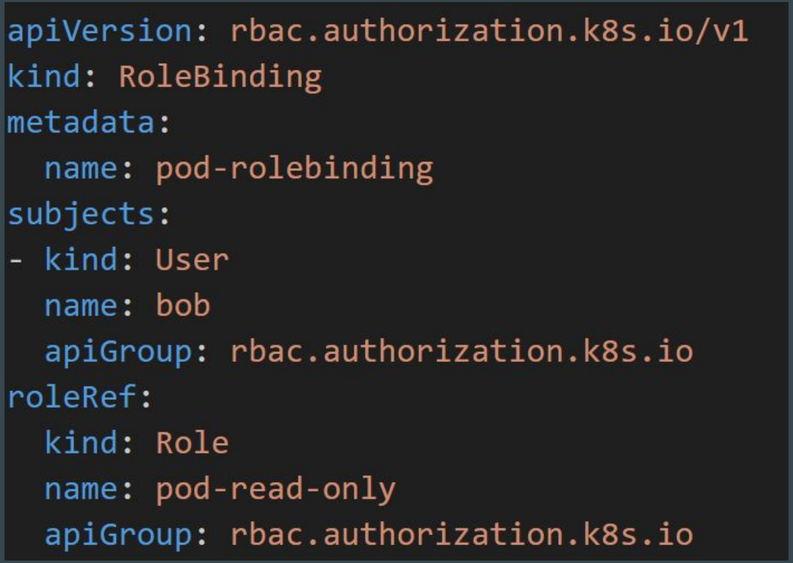
Verb specifies what actions (operations) are allowed on the specified resources.

Common Verbs	Description
get	Read a specific resource.
list	List all resources of that type.
create	Create a new resource.
delete	Modify an existing resource.
update	Remove a resource.
watch	Observe changes to a resource.

Structure - RoleBinding

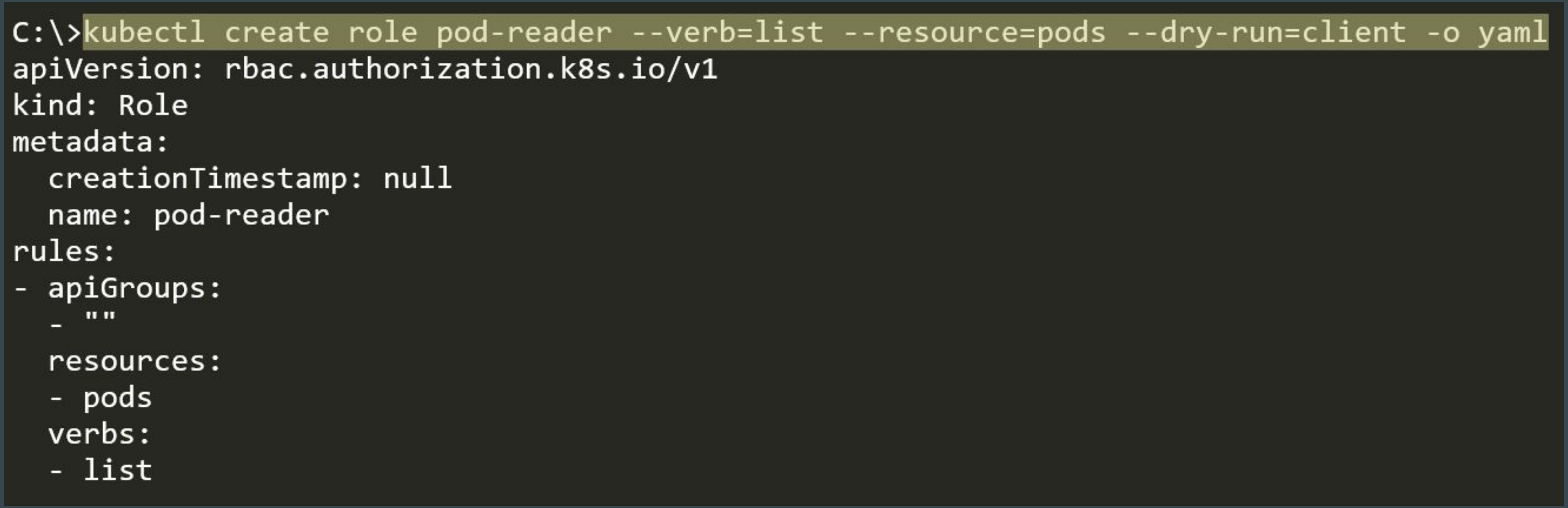
While defining RoleBinding, we have to define subjects and Role Reference.





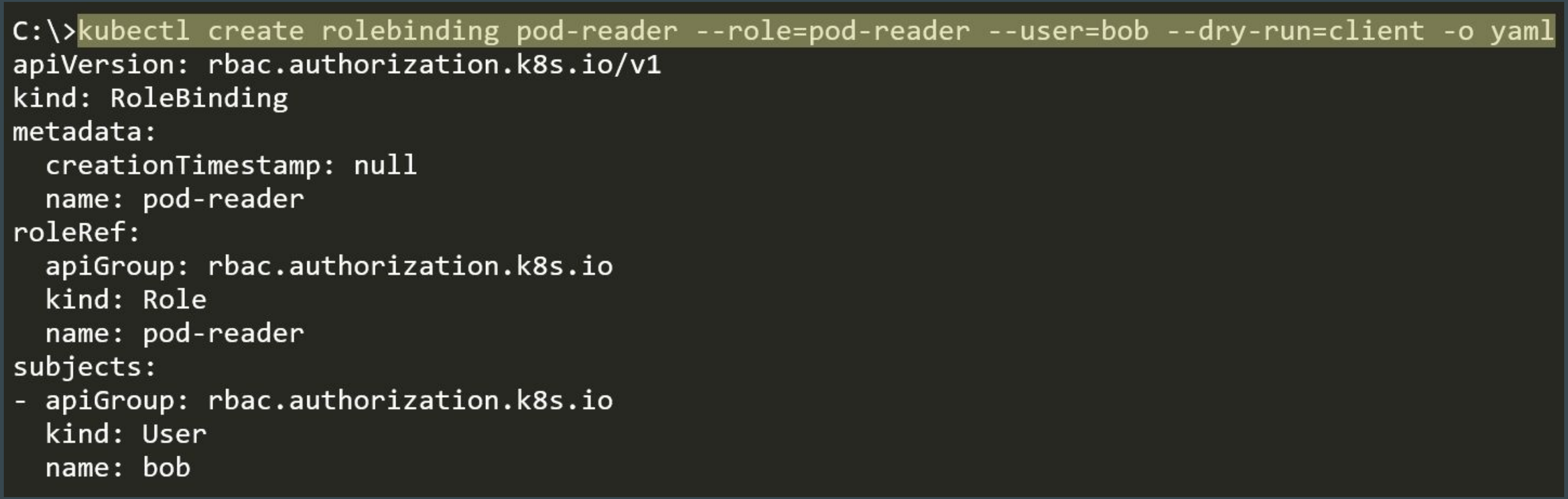
```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: pod-rolebinding
subjects:
- kind: User
  name: bob
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-read-only
  apiGroup: rbac.authorization.k8s.io
```

Generate Role Manifest File



```
C:\>kubectl create role pod-reader --verb=list --resource=pods --dry-run=client -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  creationTimestamp: null
  name: pod-reader
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - list
```

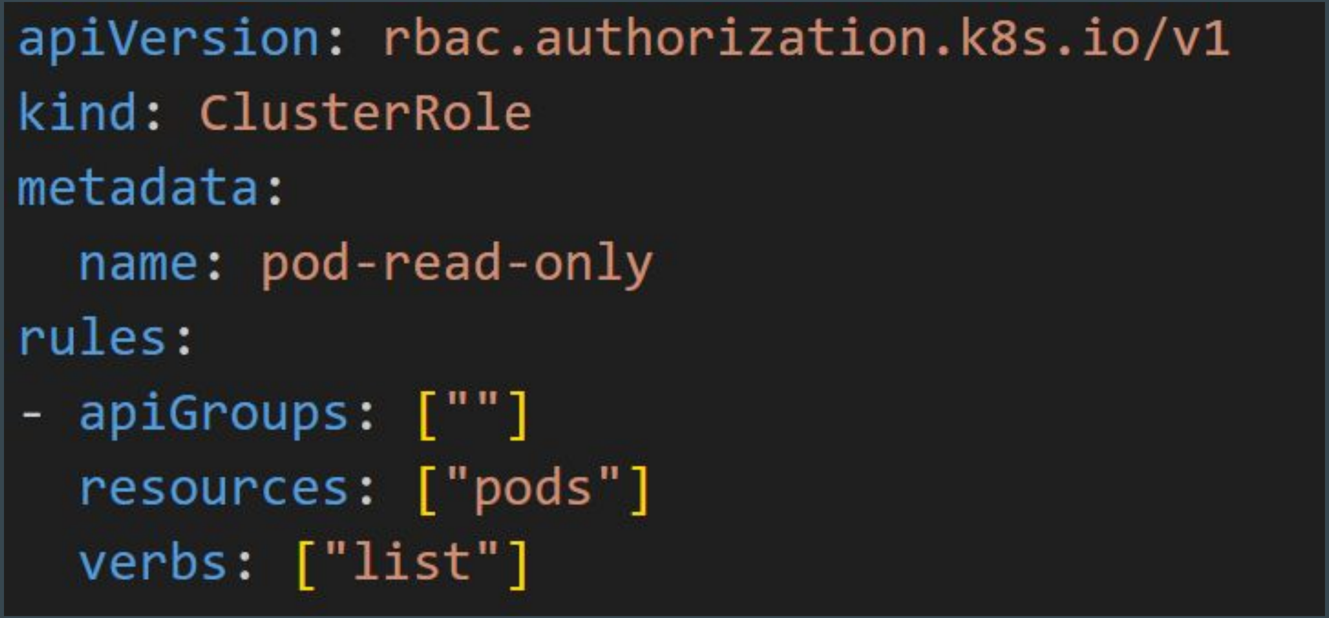
Generate Role Binding Manifest File



```
C:\>kubectl create rolebinding pod-reader --role=pod-reader --user=bob --dry-run=client -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  creationTimestamp: null
  name: pod-reader
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: pod-reader
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: bob
```

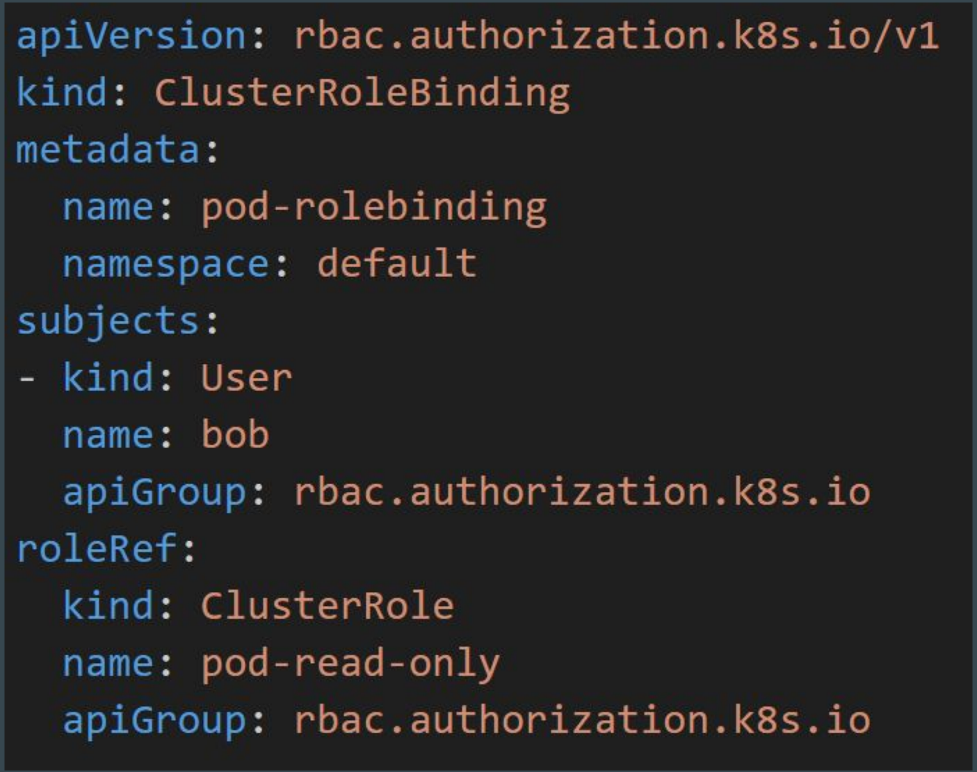
Practical - ClusterRole and ClusterRoleBinding

Structure of ClusterRole Manifest



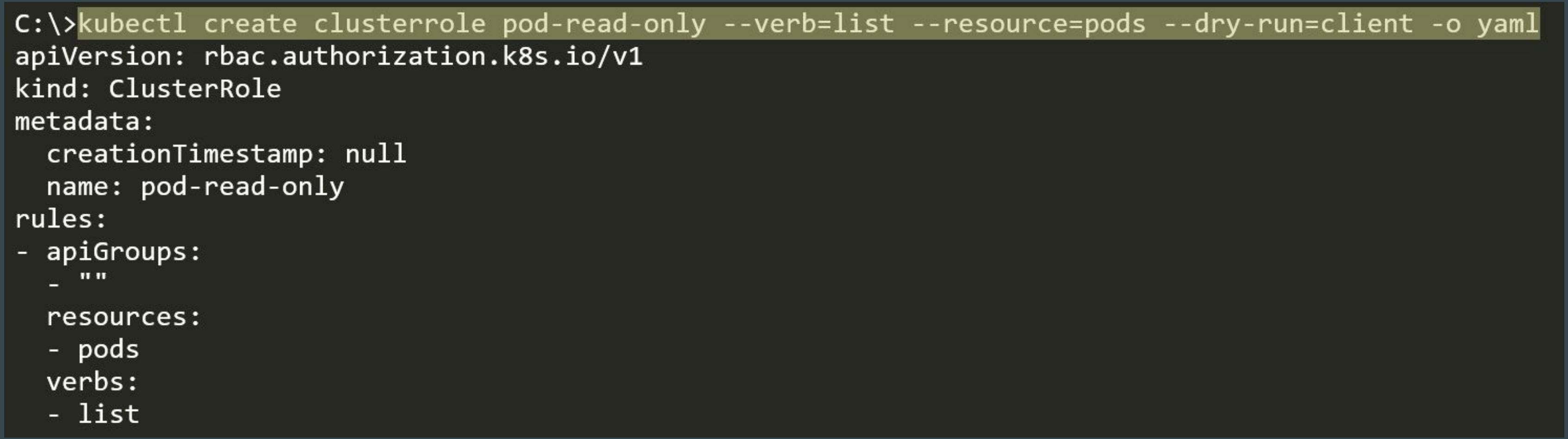
```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pod-read-only
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["list"]
```

Structure of ClusterRoleBinding Manifest



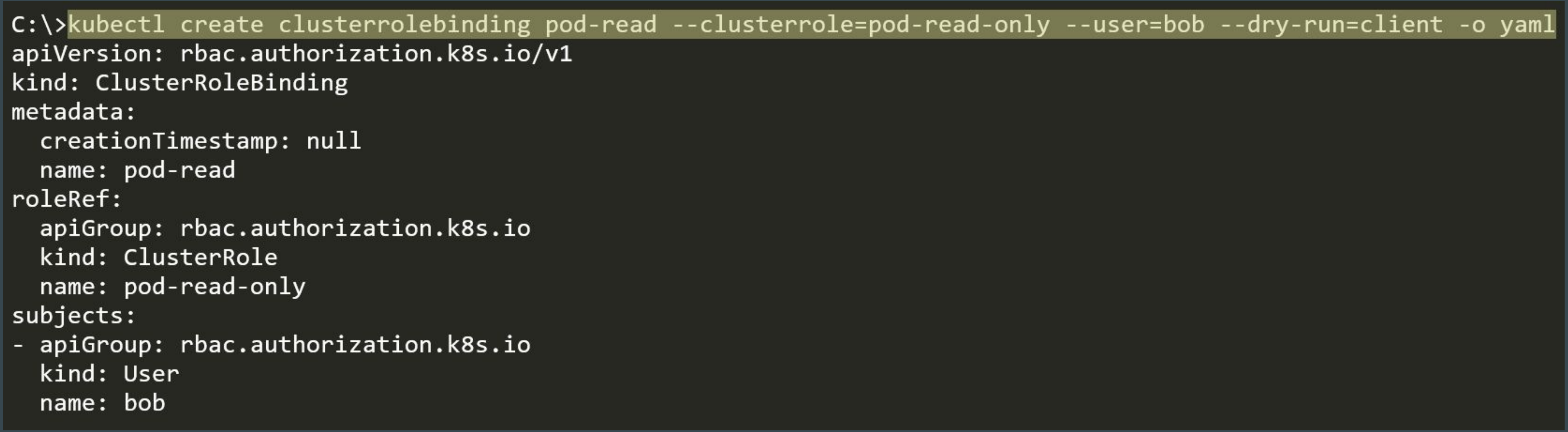
```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: pod-rolebinding
  namespace: default
subjects:
- kind: User
  name: bob
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: pod-read-only
  apiGroup: rbac.authorization.k8s.io
```

Generate ClusterRole Manifest File



```
C:\>kubectl create clusterrole pod-read-only --verb=list --resource=pods --dry-run=client -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  creationTimestamp: null
  name: pod-read-only
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - list
```

Generate ClusterRoleBinding Manifest File



```
C:\>kubectl create clusterrolebinding pod-read --clusterrole=pod-read-only --user=bob --dry-run=client -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  creationTimestamp: null
  name: pod-read
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: pod-read-only
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: bob
```