# Understand Authentication, Authorization, and Admission Control

**Elle Krout**
Principal Course Author, Pluralsight

# Authentication in Kubernetes

# Authentication

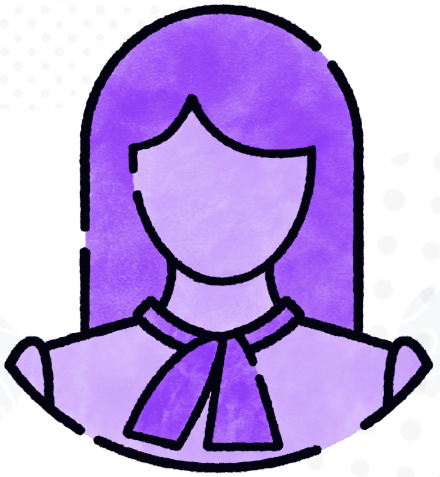
User

Request

Kubernetes API

# Authentication

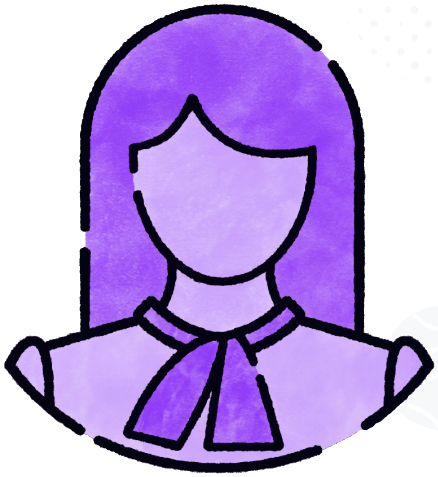
Is this user who they say they are?



User

User

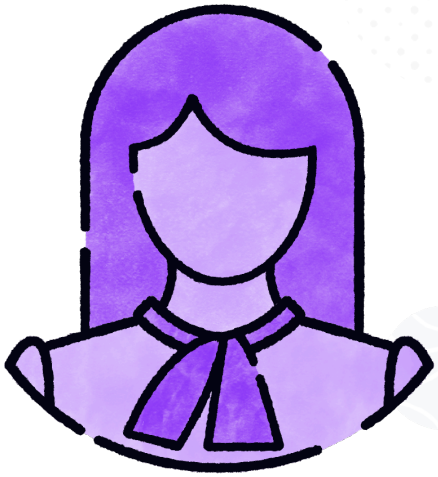
# Authentication



User

# Authentication

# Authentication



User

# Authentication

User

Request

Kubernetes API

# Authentication Methods

Client certificates

Bearer tokens

OpenID Connect providers

# Authentication Methods

Client certificates

# Client Certificates

**TLS certificates**

**Used by:**

- Users and components to verify identity
- Kuberentes controller to verify worker identity

# Client Certificates

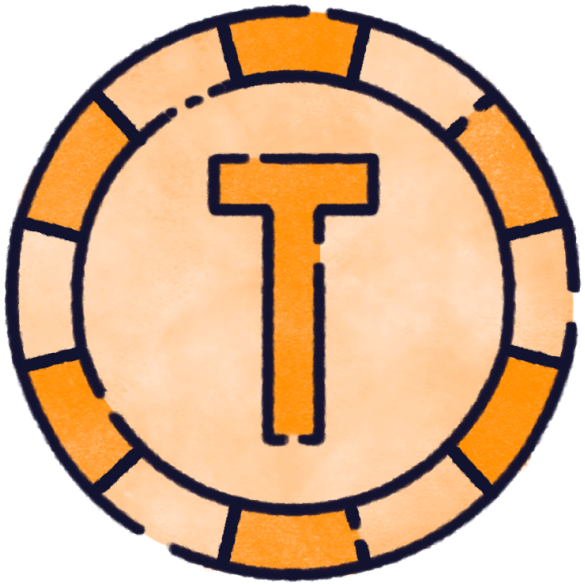
Client Certificate

Signed Certificate

Name

Organization

# Bearer Tokens



Used by service accounts

Strings passed in the the HTTP authorization header

Use in API requests

More information in “Understand Service Accounts” module

# OpenID Connect Providers

**Authentication with external identity provider**

**Ideal for:**

- Larger teams
- Teams already using SSO

**Authenticate with SSO provider, receives signed token to pass to Kubernetes**

**Not part of exam**

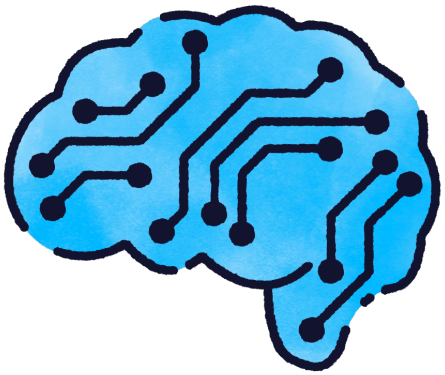
# No authentication? No interaction.

Identity is the foundation for secure and controlled access to cluster resources.

# Authorization in Kubernetes

# Does the user have permission to perform the requested action?

# Role-based Access Control



**Role**

An overview of actions that can be performed against defined components

**Rolebinding**

Assigns role to users, groups, and/or service accounts

# Role-based Access Control

# Role-based Access Control

`Role`

`ClusterRole`

`RoleBinding`

`ClusterRoleBinding`

# Create a Role

**pod-creator-role.yaml**

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: dev
  name: pod-creator
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "list", "create"]
```

# Create a Role

RBAC API Reference

**pod-creator-role.yaml**

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: dev
  name: pod-creator
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "list", "create"]
```

# Create a Role

## Identify Namespace

**pod-creator-role.yaml**

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: dev
  name: pod-creator
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "list", "create"]
```

# Create a Role

## Define Rules

**pod-creator-role.yaml**

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: dev
  name: pod-creator
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "list", "create"]
```

# Create a Role

## Define Rules: Set API Group

**pod-creator-role.yaml**

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: dev
  name: pod-creator
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "list", "create"]
```

# Create a Role

## Define Rules: Set Resources for Listed Actions

**pod-creator-role.yaml**

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: dev
  name: pod-creator
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "list", "create"]
```

# Create a Role

## Define Rules: Set Permissted Actions

**pod-creator-role.yaml**

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: dev
  name: pod-creator
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "list", "create"]
```

# Create a Cluster Role

**pod-creator-clusterrole.yaml**

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

# Create Role via Command Line

```
> kubectl create role pod-creator --verb=get,list,create --resource=pods --dry-run=client -o yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  creationTimestamp: null
  name: pod-creator
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - get
  - list
  - create
```

# Create Role via Command Line

```
> kubectl create role pod-creator --verb=get,list,create --resource=pods --
namespace=dev --dry-run=client -o yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  creationTimestamp: null
  name: pod-creator
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - get
  - list
  - create
```

# Create Cluster Role via Command Line

```
> kubectl create clusterrole pod-creator --verb=get,list,create --resource=pods --dry-run=client -o yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  creationTimestamp: null
  name: pod-creator
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - get
  - list
  - create
```

# Create a RoleBinding

**pod-creator-rolebinding.yaml**

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: bind-pod-creator
  namespace: dev
subjects:
  - kind: User
    name: bbelcher
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-creator
  apiGroup: rbac.authorization.k8s.io
```

# Create a RoleBinding

RBAC API Reference

**pod-creator-rolebinding.yaml**

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: bind-pod-creator
  namespace: dev
subjects:
  - kind: User
    name: bbelcher
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-creator
  apiGroup: rbac.authorization.k8s.io
```

# Create a RoleBinding

## Define Bound Subjects

**pod-creator-rolebinding.yaml**

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: bind-pod-creator
  namespace: dev
subjects:
  - kind: User
    name: bbelcher
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-creator
  apiGroup: rbac.authorization.k8s.io
```

# Create a RoleBinding

**Define Bound Subjects: Set Kind (`User`, `Group`, `ServiceAccount`)**

**pod-creator-rolebinding.yaml**

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: bind-pod-creator
  namespace: dev
subjects:
  - kind: User
    name: bbelcher
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-creator
  apiGroup: rbac.authorization.k8s.io
```

# Create a RoleBinding

## Define Bound Subjects

**pod-creator-rolebinding.yaml**

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: bind-pod-creator
  namespace: dev
subjects:
  - kind: User
    name: bbelcher
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-creator
  apiGroup: rbac.authorization.k8s.io
```

# Create a RoleBinding

## Define Bound Subjects

**pod-creator-rolebinding.yaml**

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: bind-pod-creator
  namespace: dev
subjects:
  - kind: User
    name: bbelcher
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-creator
  apiGroup: rbac.authorization.k8s.io
```

# Create a RoleBinding

## Define Role

**pod-creator-rolebinding.yaml**

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: bind-pod-creator
  namespace: dev
subjects:
  - kind: User
    name: bbelcher
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-creator
  apiGroup: rbac.authorization.k8s.io
```

# Create a Cluster RoleBinding

**pod-creator-clusterrolebinding.yaml**

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: bind-pod-creator
subjects:
  - kind: User
    name: bbelcher
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-creator
  apiGroup: rbac.authorization.k8s.io
```

# Create RoleBinding via Command Line

```
> kubectl create rolebinding bind-pod-creator --role=pod-creator --user=bbelcher --
namespace=dev --dry-run=client -o yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  creationTimestamp: null
  name: bind-pod-creator
  namespace: dev
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: pod-creator
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: bbelcher
```

# Create Cluster RoleBinding via Command Line

```
> kubectl create clusterrolebinding bind-pod-creator --role=pod-creator --
user=bbelcher --namespace=dev --dry-run=client -o yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  creationTimestamp: null
  name: bind-pod-creator
  namespace: dev
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: pod-creator
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: bbelcher
```

# Confirm User Can Perform Action

```
> kubectl auth can-i <verb> <resource> --as <username|group|account> --namespace
<namespace>
```

```
yes
```

# Confirm User Can Perform Action

```
> kubectl auth can-i get pods --as mr_sherbert --namespace default
```

no

# Exam Tip!

**Know how to define, bind, and validate roles and cluster roles—either through manifests or via the command line.**

# Demo: Creating a Role in Kubernetes

# Exam Scenario

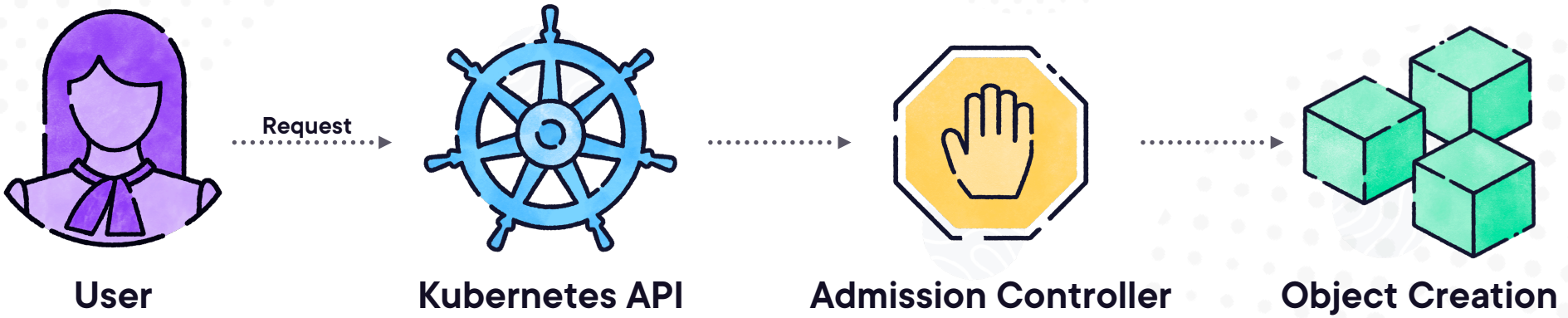
**The backend team needs RBAC access to work within the `default` namespace. Create a role that allows the team to:**

- Retrieve, view, and watch pods and deployments
- Deploy and update pods and deployments

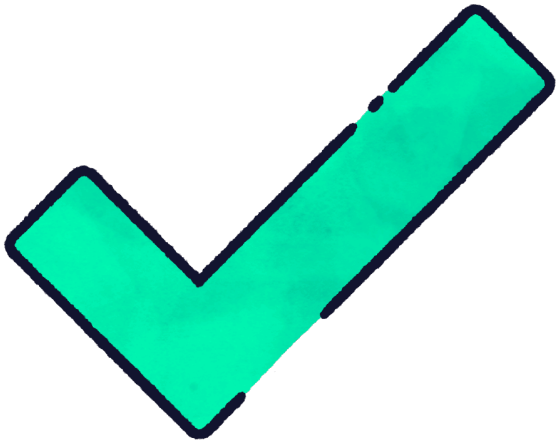
**Once the role has been created, assign it to the backend team, as well as the user `bbelcher`**

# Kubernetes Admission Control

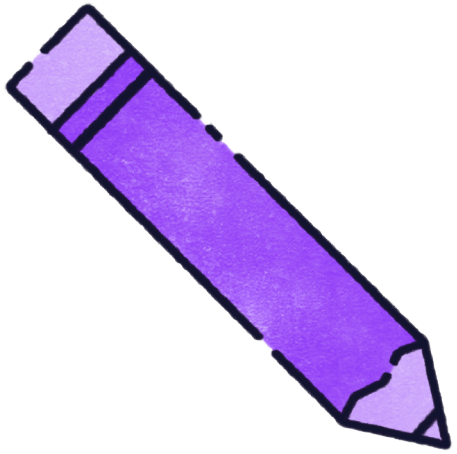
# Admission Controllers



# Types of Admission Controllers

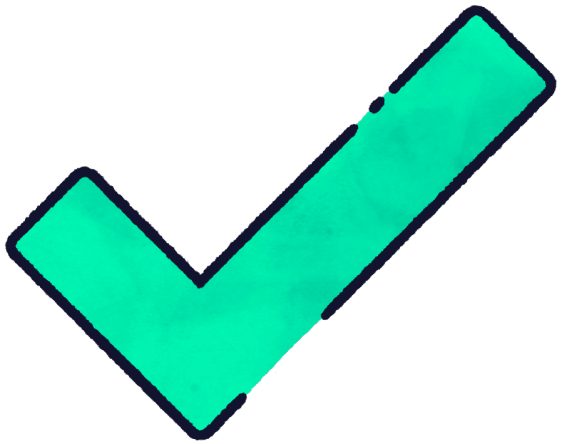


Validating



Mutating

# Validating Admission Controllers



**Checks if object meets defined criteria and either accept to rejects the request**

**Example: `EventRateLimit` Controller**

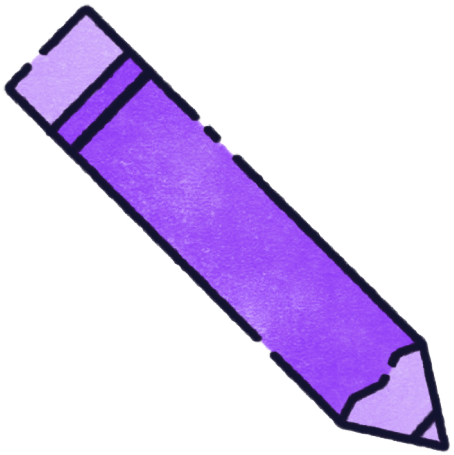
- Mitigates floods of event requests
- Requires configuration

# EventRateLimit Admission Controller Configuration

**eventratelimit-config.yaml**

```yaml
apiVersion: apiserver.config.k8s.io/v1
kind: AdmissionConfiguration
plugins:
  - name: EventRateLimit
    configuration:
      apiVersion: eventratelimit.admission.k8s.io/
v1alpha1
      kind: Configuration
      limits:
        - type: Server
          qps: 50
          burst: 100
        - type: Namespace
          qps: 10
          burst: 20
        - type: User
          qps: 5
          burst: 10
```

# Mutating Admission Controllers



**Alters object if certain criteria are not met**

**Example: `DefaultStorageClass` Controller**

- Checks if storage class is set
- Adds a default storage class if not

# DefaultStorageClass Admission Controller Configuration

**default-storage-class.yaml**

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: demo-default
  annotations:
    storageclass.kubernetes.io/is-default-class:
"true"
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

**Controllers can be both validating and mutating**

**`AlwaysPullImage` Controller:**

- Validates if image is set to aways pull
- Mutating object to ensure it is, if not

kubernetes.io/docs/reference/access-authn-authz/admission-controllers/#what-does-each-admission-controller-do

# Exam Tip!

**Will not NOT need to know how to add or create admission controllers, just how to use them.**

# Check Enabled Admission Controllers

```
> kube-apiserver -h | grep enable-admission-plugins
```

```
> sudo $EDITOR /etc/kubernetes/manifests/kube-apiserver.yaml
```

```
...
spec:
  containers:
  - command:
    - kube-apiserver
    - --admission-control-config-file=<config_file_location>
    - --advertise-address=172.31.112.111
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction,DenyServiceExternalIPs
    - --enable-bootstrap-token-auth=true
```

# Check Enabled Admission Controllers

```
> kube-apiserver -h | grep enable-admission-plugins
```

```
> sudo $EDITOR /etc/kubernetes/manifests/kube-apiserver.yaml
```

```
...
spec:
  containers:
  - command:
    - kube-apiserver
    - --admission-control-config-file=<config_file_location>
    - --advertise-address=172.31.112.111
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
```

# Exam Tip!

**Understand how admission controllers might work within the context of the tasks presented in the exam.**

# Demo: Creating a Role in Kubernetes

# Exam Scenario

In attempt to simplify namespaces, all teams are now using the same dev namespace. Remove the `frontend-dev` namespace from the cluster. Quit the command before the deletion completes

Next, deploy the service found in `internal-metrics.yaml`. Resolve any errors