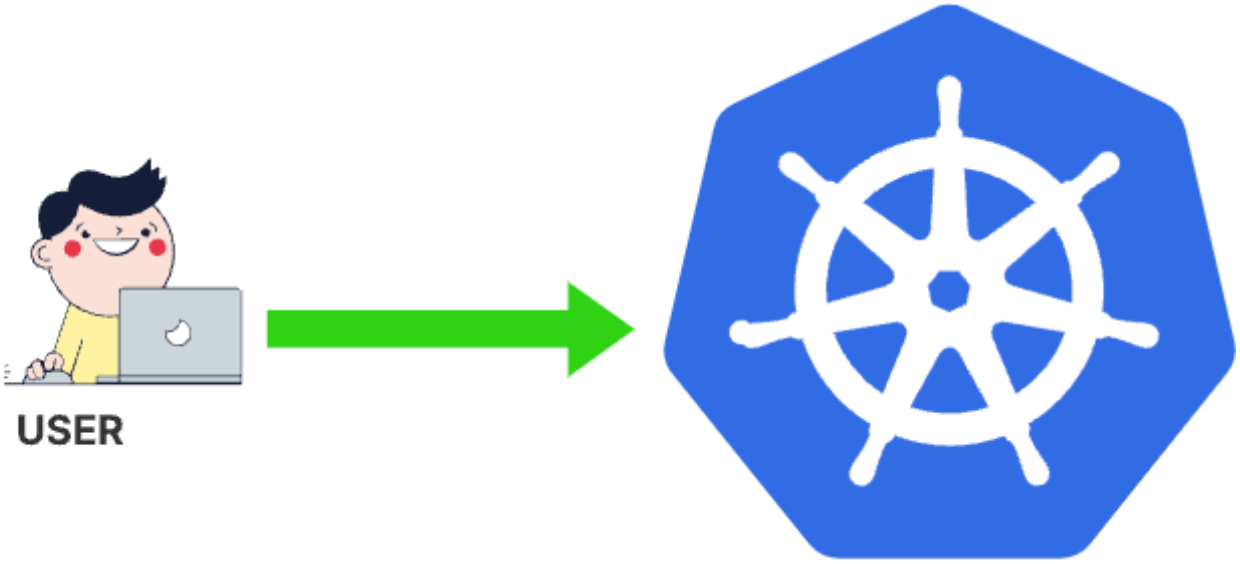


How to Create a User in a Kubernetes Cluster and Grant Access

Hakan Bayraktar

Follow

~4 min read · November 17, 2023 (Updated: November 18, 2023) · Free: Yes

In this detailed guide, we'll illustrate the steps required to create a user, generate necessary certificates, and configure access using a kubeconfig file within a Kubernetes cluster.

Step 1: Generating a Key Pair and Certificate Signing Request (CSR)

First, let's generate a key pair and a Certificate Signing Request (CSR) using OpenSSL:

Copy

```
openssl genrsa -out developer.key 2048
openssl req -new -key developer.key -out developer.csr -subj "/CN=developer"
```

```
root@hakan-yeni:~/rbac# openssl genrsa -out developer.key 2048
root@hakan-yeni:~/rbac# openssl req -new -key developer.key -out developer.csr -subj "/CN=developer"
```

Now, let's create create a CSR YAML file named "`csr_template.yaml`" to submit to Kubernetes:

`csr_template.yaml`

Copy

```
cat <<EOF > csr_template.yaml
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: developer-csr
spec:
  request: <Base64_encoded_CSR>
  signerName: kubernetes.io/kube-apiserver-client
  usages:
  - client auth
EOF
```

Replace `<Base64_encoded_CSR>` with the Base64-encoded content of the `developer.csr` file.

Generate the CSR content in Base64 and create the YAML file:

Copy

```
CSR_CONTENT=$(cat developer.csr | base64 | tr -d '\n')
sed "s|<Base64_encoded_CSR>|$CSR_CONTENT|" csr_template.yaml > developer_cs
```

Apply the CSR YAML file to Kubernetes:

Copy

```
kubectl create -f developer_csr.yaml
```

```
root@hakan-yeni:~/rbac# cat <<EOF > csr_template.yaml
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: developer-csr
spec:
  request: <Base64_encoded_CSR>
  signerName: kubernetes.io/kube-apiserver-client
  usages:
  - client auth
EOF
root@hakan-yeni:~/rbac# CSR_CONTENT=$(cat developer.csr | base64 | tr -d '\n')
sed "s|<Base64_encoded_CSR>|$CSR_CONTENT|" csr_template.yaml > developer_csr.yaml
root@hakan-yeni:~/rbac# kubectl create -f developer_csr.yaml
certificatesigningrequest.certificates.k8s.io/developer-csr created
root@hakan-yeni:~/rbac#
```

Approve the CSR and retrieve the approved certificate:

Copy

```
kubectl get csr
kubectl certificate approve developer-csr
kubectl get csr developer-csr -o jsonpath='{.status.certificate}' | base64
kubectl get csr
```

```
root@hakan-yeni:~/rbac# kubectl get csr
NAME            AGE   SIGNERNAME                            REQUESTOR          REQUESTEDDURATION   CONDITION
developer-csr   66s   kubernetes.io/kube-apiserver-client   kubernetes-admin   <none>              Pending
root@hakan-yeni:~/rbac# kubectl certificate approve developer-csr
certificatesigningrequest.certificates.k8s.io/developer-csr approved
root@hakan-yeni:~/rbac# kubectl get csr developer-csr -o jsonpath='{.status.certificate}' | base64 --decode > developer.crt
root@hakan-yeni:~/rbac# kubectl get csr
NAME            AGE    SIGNERNAME                            REQUESTOR          REQUESTEDDURATION   CONDITION
developer-csr   106s   kubernetes.io/kube-apiserver-client   kubernetes-admin   <none>              Approved,Issued
root@hakan-yeni:~/rbac#
```

Step 2: Generate and Configure a kubeconfig File

To access the Kubernetes cluster, it's essential to generate a configuration file tailored for the 'developer' user. This file needs to encompass critical information, including the Kubernetes API access specifics, the Cluster CA

certificate, as well as the 'developer' user's certificate and context name. Initially, we'll generate the kubeconfig file specifically for the 'developer' user.

Configure the kubeconfig file:

We need to modify below the command according to our cluster-specific information **to Set Cluster Configuration:**

kubectl config set-cluster kubernetes — server=**https://<Kubernetes_API_server_endpoint>:<port>** — certificate-authority=**<Base64_encoded_CA_certificate>** — embed-certs=true — kubeconfig=developer.kubeconfig

> *Replace* ***<Kubernetes_API_server_endpoint>*** *with the address of the Kubernetes API server and* ***<port>*** *with the corresponding port number. Also, replace* ***<Base64_encoded_CA_certificate>*** *with the file path of the CA certificate in Base64 encoding.*

First, we need to locate the cluster's Kubernetes API access details and the Cluster CA certificate:

Copy

```
kubectl config view
ls /etc/kubernetes/pki/
```

```
root@hakan-yeni:~# kubectl get csr
No resources found
root@hakan-yeni:~# kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://104.248.28.87:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: DATA+OMITTED
    client-key-data: DATA+OMITTED
root@hakan-yeni:~# ls /etc/kubernetes/pki/
apiserver-etcd-client.crt     apiserver-kubelet-client.key  ca.crt   front-proxy-ca.crt      front-proxy-client.key
apiserver-etcd-client.key     apiserver.crt                 ca.key   front-proxy-ca.key      sa.key
apiserver-kubelet-client.crt  apiserver.key                 etcd     front-proxy-client.crt  sa.pub
```

I changed the command above according our cluster information.

Copy

```
# Set Cluster Configuration:
kubectl config set-cluster kubernetes --server=https://104.248.28.87:6443 -
```

Copy

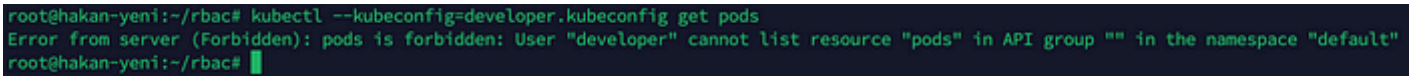
```
# Set Credentials for Developer:
kubectl config set-credentials developer --client-certificate=developer.crt
# Set Developer Context:
kubectl config set-context developer-context --cluster=kubernetes --namespa
# Use Developer Context:
kubectl config use-context developer-context --kubeconfig=developer.kubecon
```

```
root@hakan-yeni:~/rbac# kubectl config set-cluster kubernetes --server=https://104.248.28.87:6443 --certificate-authority=/etc/kubernetes/pki/ca.crt --e
mbed-certs=true --kubeconfig=developer.kubeconfig
Cluster "kubernetes" set.
root@hakan-yeni:~/rbac# kubectl config set-credentials developer --client-certificate=developer.crt --client-key=developer.key --embed-certs=true --kube
config=developer.kubeconfig
User "developer" set.
root@hakan-yeni:~/rbac# kubectl config set-context developer-context --cluster=kubernetes --namespace=default --user=developer --kubeconfig=developer.ku
beconfig
Context "developer-context" created.
```

Verify the kubeconfig file's configuration:

Copy

```
kubectl --kubeconfig=developer.kubeconfig get pods
```



We logged into the cluster with the 'Developer' user and attempted to list the pods in the 'default' namespace. However, due to the lack of necessary permissions for the 'Developer' user, we couldn't retrieve the list of pods. Below, you can find how to grant the required permissions to this user.

Step 3: Assign Roles and Bindings for the Developer User

Create and apply roles and role bindings for the developer user:

developer-cluster-role.yaml

Copy

```
cat <<EOF > developer-cluster-role.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: developer-role
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["*"]
  verbs: ["*"]
EOF
```

developer-role-binding.yaml

Copy

```
cat <<EOF > developer-role-binding.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: developer-binding
  namespace: default
subjects:
- kind: User
  name: developer
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: developer-role
  apiGroup: rbac.authorization.k8s.io
EOF
```

Apply the roles and role bindings:

Copy

```
kubectl apply -f developer-cluster-role.yaml -f developer-role-binding.yaml
```

```
root@hakan-yeni:~/rbac# cat <<EOF > developer-cluster-role.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: developer-role
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["*"]
  verbs: ["*"]
EOF
root@hakan-yeni:~/rbac# cat <<EOF > developer-role-binding.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: developer-binding
  namespace: default
subjects:
- kind: User
  name: developer
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: developer-role
  apiGroup: rbac.authorization.k8s.io
EOF
root@hakan-yeni:~/rbac# kubectl apply -f developer-cluster-role.yaml -f developer-role-binding.yaml
clusterrole.rbac.authorization.k8s.io/developer-role created
rolebinding.rbac.authorization.k8s.io/developer-binding created
root@hakan-yeni:~/rbac#
```

Step 4: Verify developer User Rights

You can run the following commands to check the permissions assigned to the 'developer' user for accessing the Kubernetes cluster resources.

Copy

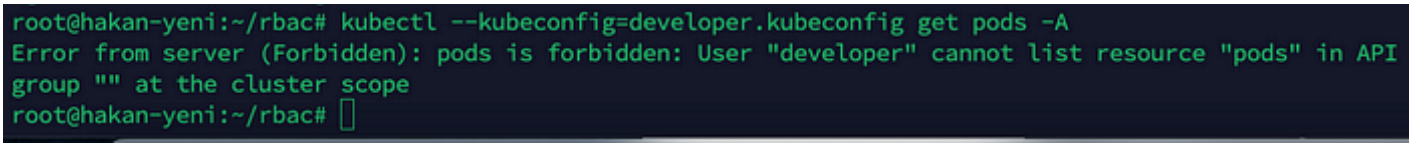
```
kubectl --kubeconfig=developer.kubeconfig get pods
kubectl --kubeconfig=developer.kubeconfig run nginx --image=nginx
kubectl --kubeconfig=developer.kubeconfig get pods
```

```
root@hakan-yeni:~/rbac# kubectl --kubeconfig=developer.kubeconfig get pods
No resources found in default namespace.
root@hakan-yeni:~/rbac# kubectl --kubeconfig=developer.kubeconfig run nginx --image=nginx
pod/nginx created
root@hakan-yeni:~/rbac# kubectl --kubeconfig=developer.kubeconfig get pods
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          4s
```

This confirms that the developer user has appropriate access to pods in the default namespace.

Copy

```
kubectl --kubeconfig=developer.kubeconfig get pods -A
```



We couldn't retrieve the information about pods across all namespaces. This limitation occurred because the permissions granted to the developer user are only applicable to the 'default' namespace.

Note: You can find more information about RBAC Authorization and different user role-related details at his link: https://kubernetes.io/docs/reference/access-authn-authz/rbac/

#kubernetes-cluster #rbac-access-control #kubeconfig #kubernetes