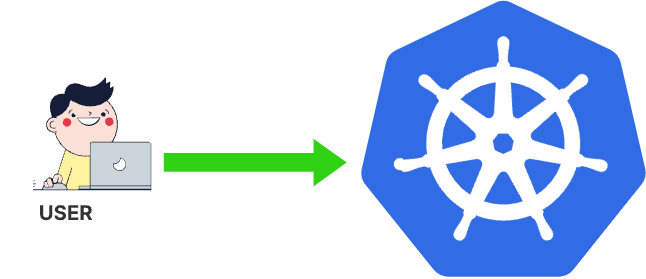
**How to Create a User in a Kubernetes Cluster and Grant Access**



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:

openssl genrsa -out developer.key 2048  
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**

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:

CSR\_CONTENT=$(cat developer.csr | base64 | tr -d '\n')  
sed "s|<Base64\_encoded\_CSR>|$CSR\_CONTENT|" csr\_template.yaml > developer\_csr.yaml

Apply the CSR YAML file to Kubernetes:

kubectl create -f developer\_csr.yaml



Approve the CSR and retrieve the approved certificate:

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



**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:**

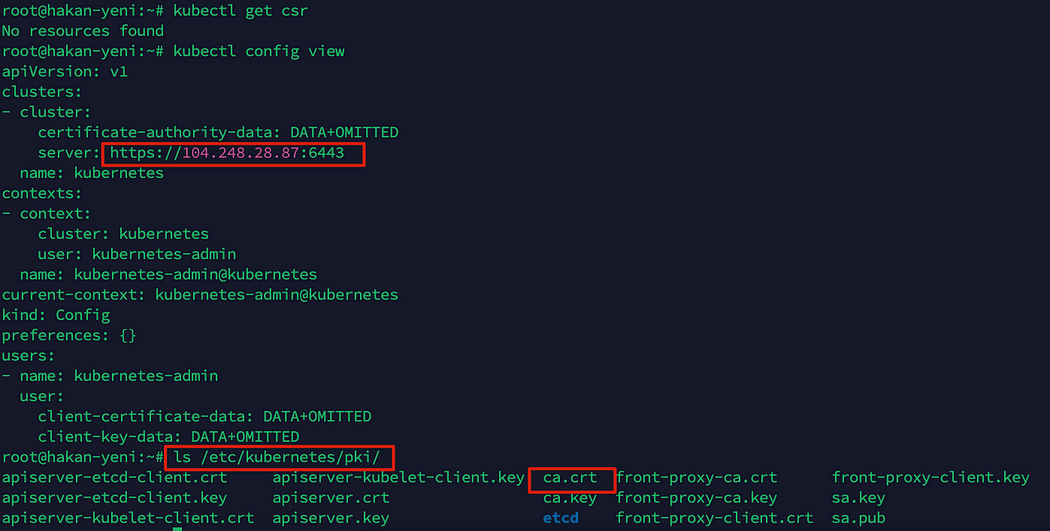
We need to modify below the command according to our cluster-specific information**to Set Cluster Configuration**:

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:

kubectl config view  
ls /etc/kubernetes/pki/



I changed the command above according our cluster information.

# Set Cluster Configuration:  
kubectl config set-cluster kubernetes --server=https://104.248.28.87:6443 --certificate-authority=/etc/kubernetes/pki/ca.crt --embed-certs=true --kubeconfig=developer.kubeconfig

# Set Credentials for Developer:  
kubectl config set-credentials developer --client-certificate=developer.crt --client-key=developer.key --embed-certs=true --kubeconfig=developer.kubeconfig  
# Set Developer Context:   
kubectl config set-context developer-context --cluster=kubernetes --namespace=default --user=developer --kubeconfig=developer.kubeconfig  
# Use Developer Context:  
kubectl config use-context developer-context --kubeconfig=developer.kubeconfig



Verify the kubeconfig file’s configuration:

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**

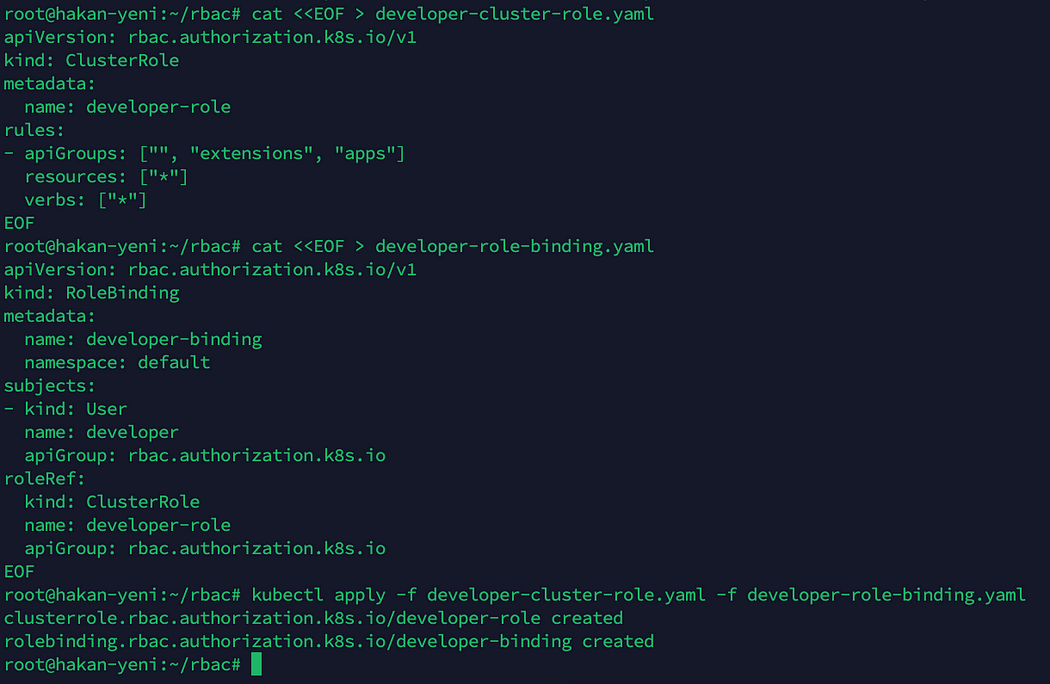
cat <<EOF > developer-cluster-role.yaml  
apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRole  
metadata:  
 name: developer-role  
rules:  
- apiGroups: ["", "extensions", "apps"]  
 resources: ["\*"]  
 verbs: ["\*"]  
EOF

**developer-role-binding.yaml**

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:

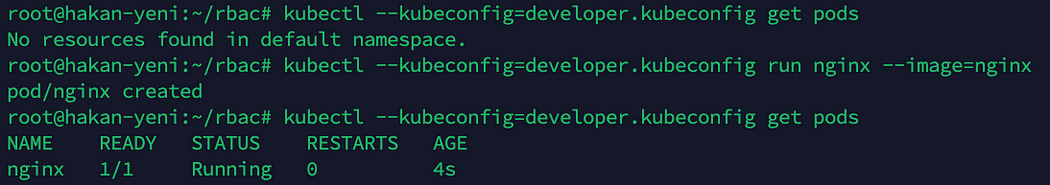
kubectl apply -f developer-cluster-role.yaml -f developer-role-binding.yaml



**Step 4: Verify developer User Rights**

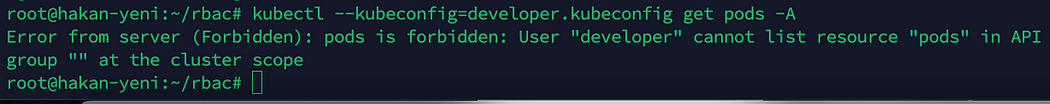
You can run the following commands to check the permissions assigned to the ‘developer’ user for accessing the Kubernetes cluster resources.

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



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

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/>