**A Step-by-Step Guide to Creating Users in Kubernetes**

Before we begin, ensuring that the secret file names match the user’s name will save your time a lot. e.g *john.key john.csr john.crt john.pem* etc.

Content Outline —

1. Generate certificates for the user.
2. Create a certificate signing request (CSR).
3. Sign the certificate using the cluster certificate authority.
4. Create a configuration specific to the user.
5. Add RBAC rules for the user or their group.

**1. Generate certificates for the user.**

Following command will generates an Ed25519 private key using OpenSSL.

openssl genpkey -out john.key -algorithm Ed25519

“*Algorithm Ed25519 not found*” If you find this error simply upgrade your openssl.

cat john.key   
  
-----BEGIN PRIVATE KEY-----  
MC4CAQAwBQYDK2VwBCIEIBDWF4uAgPnmVQ9meMtjG4cMomoq8qpfyfOZ14Ef8pYq  
-----END PRIVATE KEY-----

After executing this command, OpenSSL will generate an Ed25519(which is a modern elliptic curve cryptography algorithm) private key and save it in the “john.key” file. Keep in mind that it’s crucial to protect this private key since it grants access to sensitive information or operations.

openssl req -new -key john.key -out john.csr -subj "/CN=john,/O=edit"

In this example, the Common Name (CN) is set to “john” and the Organization (O) is set to “edit”.

**2. Create a certificate signing request (CSR).**

For request we have to obtain the base64-encoded representation of the contents of the “john.csr” file without newline characters. This encoding is often useful for transmitting or storing binary data in a text-based format.

cat john.csr | base64 | tr -d "\n"  
  
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSBSRVFVRVNULS0tLS0KTUlHZU1GSUNBUUF3SHpFT01Bd0dBMVVFQXd3RmFtOW9iaXd4RFRBTEJnTlZCQW9NQkdWa2FYUXdLakFGQmdNcgpaWEFESVFEbk9nYXEzSEcyMkw1dFROZ0JaQ1lHYkJiSjdXQ05BTmE2aWhsNll0NWFkNkFBTUFVR0F5dGxjQU5CCkFPeVJaZlpKY2c3eE90eHBjUmFqZmlySk9WcVkvaE9CQldneEJubERFek4rbXpJRm12MkU5czNoaXBQZjBOYk8KMkNrR0pNR0NhOXJabStVRHowelhqZ009Ci0tLS0tRU5EIENFUlRJRklDQVRFIFJFUVVFU1QtLS0tLQo=

Then we will copy a CertificateSigningReques template from k8 [docs](https://kubernetes.io/docs/reference/access-authn-authz/certificate-signing-requests/#create-certificatessigningrequest). Simply copy the output from previous command & replace in .spec.request.

cat <<EOF | kubectl apply -f -  
apiVersion: certificates.k8s.io/v1  
kind: CertificateSigningRequest  
metadata:  
 name: john  
spec:  
 request: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSBSRVFVRVNULS0tLS0KTUlHZU1GSUNBUUF3SHpFT01Bd0dBMVVFQXd3RmFtOW9iaXd4RFRBTEJnTlZCQW9NQkdWa2FYUXdLakFGQmdNcgpaWEFESVFEbk9nYXEzSEcyMkw1dFROZ0JaQ1lHYkJiSjdXQ05BTmE2aWhsNll0NWFkNkFBTUFVR0F5dGxjQU5CCkFPeVJaZlpKY2c3eE90eHBjUmFqZmlySk9WcVkvaE9CQldneEJubERFek4rbXpJRm12MkU5czNoaXBQZjBOYk8KMkNrR0pNR0NhOXJabStVRHowelhqZ009Ci0tLS0tRU5EIENFUlRJRklDQVRFIFJFUVVFU1QtLS0tLQo=  
 signerName: kubernetes.io/kube-apiserver-client  
 expirationSeconds: 86400 # one day  
 usages:  
 - client auth  
EOF  
  
certificatesigningrequest.certificates.k8s.io/john created

**3. Sign the certificate using the cluster certificate authority.**

By executing following command, you authorize the issuance of a certificate associated with the CSR, allowing the user to use the certificate for authentication and access within the Kubernetes cluster.

kubectl certificate approve john  
  
certificatesigningrequest.certificates.k8s.io/john approved

We can double check our user (john’s) CSR status.

kubectl describe csr/john   
Name: john  
Labels: <none>  
Annotations: kubectl.kubernetes.io/last-applied-configuration={"apiVersion":"certificates.k8s.io/v1","kind":"CertificateSigningRequest","metadata":{"annotations":{},"name":"john"},"spec":{"expirationSeconds":86400,"request":"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSBSRVFVRVNULS0tLS0KTUlHZU1GSUNBUUF3SHpFT01Bd0dBMVVFQXd3RmFtOW9iaXd4RFRBTEJnTlZCQW9NQkdWa2FYUXdLakFGQmdNcgpaWEFESVFEbk9nYXEzSEcyMkw1dFROZ0JaQ1lHYkJiSjdXQ05BTmE2aWhsNll0NWFkNkFBTUFVR0F5dGxjQU5CCkFPeVJaZlpKY2c3eE90eHBjUmFqZmlySk9WcVkvaE9CQldneEJubERFek4rbXpJRm12MkU5czNoaXBQZjBOYk8KMkNrR0pNR0NhOXJabStVRHowelhqZ009Ci0tLS0tRU5EIENFUlRJRklDQVRFIFJFUVVFU1QtLS0tLQo=","signerName":"kubernetes.io/kube-apiserver-client","usages":["client auth"]}}  
  
CreationTimestamp: Wed, 21 Jun 2023 14:28:59 +0000  
Requesting User: system:admin  
Signer: kubernetes.io/kube-apiserver-client  
Requested Duration: 24h  
Status: Approved,Issued  
Subject:  
 Common Name: john,  
 Serial Number:   
 Organization: edit  
Events: <none>

Please note that in production env, you’ll need the appropriate permissions and role-based access control (RBAC) privileges to execute this command successfully. But we’re here for short & quick proto, aren’t we?

**4. Create a configuration specific to the user.**

kubectl get csr/john -o jsonpath="{.status.certificate}" | base64 -d > john.crt  
  
  
cp ~/.kube/config john-kube-config #always backup  
  
  
kubectl config get-clusters  
  
NAME  
default

Now we **extracts the certificate** from a CertificateSigningRequest (CSR) named “john” using kubectl and jsonpath. The certificate is base64-decoded and saved in a file named "john.crt". Keep in mind that we only have one cluster in this case named default .

kubectl --kubeconfig john-kube-config config set-credentials john --client-key john.key --client-certificate john.csr --embed-certs=true  
  
User "john" set.

To **set credentials** for a user named “john” using a Kubernetes configuration file named “john-kube-config”. It specifies the client key, client certificate, and enables embedding of the certificates.

kubectl --kubeconfig john-kube-config config set-context john --cluster default --user john  
  
Context "john" created.

To **set a context** named “john” in a Kubernetes configuration file named “john-kube-config”. It specifies the cluster default and user john to be associated with this context.

**5. Add RBAC rules for the user or their group.**

Create a Role for John to manage pods in the default namespace:

kubectl create role pod-manager --verb=create,list,get --resource=pods --namespace=default

Bind the Role with a ClusterRoleBinding:

kubectl create clusterrolebinding john-pod-manager — clusterrole=pod-manager — user=john

Verify authorization using kubectl auth can-i:

kubectl auth can-i create deployments --namespace=default --as=john  
no   
  
kubectl auth can-i create secrets --namespace=default --as=john  
no  
  
kubectl auth can-i get pods --namespace=default --as=john  
yes

Good work, we successfully create pod-manager called John. Inspire by [this repo](https://github.com/brendandburns/kubernetes-adduser) from Brendan Burns.

We still have more problems to tackle —

**Key Distribution**: Distributing cryptographic keys securely is essential to maintain the confidentiality and integrity of sensitive information. How do we share it? mailed? Drive or Dropbox it?

**Key Expiration**: Setting expiration dates for cryptographic keys is crucial for maintaining security. Regularly rotating keys helps mitigate the risk of compromised keys over time. We want the user to update the key easily but not so easily for unauthorized one.

**Key Security**: Protecting cryptographic keys is of utmost importance. It is important to consider industry best practices and relevant standards. Because in security context everything is vulnerable.

**Key Breaches**: If a key is compromised, it can lead to severe security breaches. Do we have incident response plan, or key revocation and replacement.

While the security landscape is ever-evolving and vulnerabilities can emerge, comprehensive security practices, risk assessments, and a proactive approach can significantly mitigate risks and enhance overall security. Hope it’s useful and have a good one.