Creating users and groups using x509 certificates

**Creating Kubernetes Users and Groups**

Generate the user's private key

1. mkdir -p ~/.kube/users
2. cd ~/.kube/users
4. openssl genrsa -out maya.key 2048
5. openssl genrsa -out kim.key 2048
6. openssl genrsa -out yono.key 2048

[sample Output]

1. openssl genrsa -out maya.key 2048
2. Generating RSA private key, 2048 bit long modulus
3. .............................................................+++
4. .........................+++
5. e is 65537 (0x10001)

Lets now create a **Certification Signing Request (CSR)** for each of the users. When you generate the csr make sure you also provide

* CN: This will be set as username
* O: Org name. This is actually used as a **group** by kubernetes while authenticating/authorizing users. You could add as many as you need

e.g.

1. openssl req -new -key maya.key -out maya.csr -subj "/CN=maya/O=ops/O=example.org"
2. openssl req -new -key kim.key -out kim.csr -subj "/CN=kim/O=dev/O=example.org"
3. openssl req -new -key yono.key -out yono.csr -subj "/CN=yono/O=interns/O=example.org"

In order to be deemed authentic, these CSRs need to be signed by the **Certification Authority (CA)** which in this case is Kubernetes Master. You need access to the folllwing files on kubernetes master.

* Certificate : ca.crt (kubeadm) or ca.key (kubespray)
* Pricate Key : ca.key (kubeadm) or ca-key.pem (kubespray)

You would typically find it at one of the following paths

* /etc/kubernetes/pki (kubeadm)
* /etc/kubernetes/ssl (kubespray)

To verify which one is your cert and which one is key, use the following command,

1. $ file ca.pem
2. ca.pem: PEM certificate

5. $ file ca-key.pem
6. ca-key.pem: PEM RSA private key

Once signed, .csr files with added signatures become the certificates that could be used to authenticate.

You could either

* move the crt files to k8s master, sign and download
* copy over the CA certs and keys to your management node and use it to sign. Make sure to keep your CA related files secure.

In the example here, I have already downloaded **ca.pem** and **ca-key.pem** to my management workstation, which are used to sign the CSRs.

Assuming all the files are in the same directory, sign the CSR as,

1. openssl x509 -req -CA ca.pem -CAkey ca-key.pem -CAcreateserial -days 730 -in maya.csr -out maya.crt
3. openssl x509 -req -CA ca.pem -CAkey ca-key.pem -CAcreateserial -days 730 -in kim.csr -out kim.crt
5. openssl x509 -req -CA ca.pem -CAkey ca-key.pem -CAcreateserial -days 730 -in yono.csr -out yono.crt

**Setting up User configs with kubectl**

In order to configure the users that you created above, following steps need to be performed with kubectl

* Add credentials in the configurations
* Set context to login as a user to a cluster
* Switch context in order to assume the user's identity while working with the cluster

to add credentials,

1. kubectl config set-credentials maya --client-certificate=/absolute/path/to/maya.crt --client-key=/absolute/path/to/maya.key
3. kubectl config set-credentials kim --client-certificate=/absolute/path/to/kim.crt --client-key=~/.kube/users/kim.key
5. kubectl config set-credentials yono --client-certificate=/absolute/path/to/yono.crt --client-key=~/.kube/users/yono.key

**where,**

* Replace /absolute/path/to/ with the path to these files.
  + invalid : ~/.kube/users/yono.crt
  + valid : /home/xyz/.kube/users/yono.crt

And proceed to set/create contexts (user@cluster). If you are not sure whats the cluster name, use the following command to find,

1. kubectl config get-contexts

**[sample output]**

1. CURRENT NAME CLUSTER AUTHINFO NAMESPACE
2. admin-prod prod admin-cluster.local instavote
3. admin-cluster4 cluster4 admin-cluster4 instavote
4. \* kubernetes-admin@kubernetes kubernetes kubernetes-admin instavote

where, **prod**, **cluster4** and **kubernetes** are cluster names.

To set context for **prod** cluster,

1. kubectl config set-context maya-prod --cluster=prod --user=maya --namespace=instavote
3. kubectl config set-context kim-prod --cluster=prod --user=kim --namespace=instavote
5. kubectl config set-context yono-prod --cluster=prod --user=yono --namespace=instavote

Where,

* maya-prod : name of the context
* prod : name of the kubernetes cluster you set while creating it
* maya : user you created and configured above to connect to the cluster

You could verify the configs with

1. kubectl config get-contexts
3. CURRENT NAME CLUSTER AUTHINFO NAMESPACE
4. \* admin-prod prod admin-prod
5. kim-prod prod kim
6. maya-prod prod maya
7. yono-prod prod yono

and

1. kubectl config view
3. apiVersion: v1
4. clusters:
5. - cluster:
6. certificate-authority-data: REDACTED
7. server: https://128.199.248.240:6443
8. name: prod
9. contexts:
10. - context:
11. cluster: prod
12. user: admin-prod
13. name: admin-prod
14. - context:
15. cluster: prod
16. user: kim
17. name: kim-prod
18. - context:
19. cluster: prod
20. user: maya
21. name: maya-prod
22. - context:
23. cluster: prod
24. user: yono
25. name: yono-prod
26. current-context: admin-prod
27. kind: Config
28. preferences: {}
29. users:
30. - name: admin-prod
31. user:
32. client-certificate-data: REDACTED
33. client-key-data: REDACTED
34. - name: maya
35. user:
36. client-certificate: users/~/.kube/users/maya.crt
37. client-key: users/~/.kube/users/maya.key

*You could assume the identity of user****yono****and connect to the****prod****cluster as,*

1. kubectl config use-context yono-prod
3. kubectl config get-contexts
5. CURRENT NAME CLUSTER AUTHINFO NAMESPACE
6. admin-prod prod admin-prod
7. kim-prod prod kim
8. maya-prod prod maya
9. \* yono-prod prod yono

And then try running any command as,

1. kubectl get pods

**Alternately, if you are a admin user, you could impersonate a user and run a command with that literally using --as option**

1. kubectl config use-context admin-prod
2. kubectl get pods --as yono

*[Sample Output]*

1. No resources found.
2. Error from server (Forbidden): pods is forbidden: User "yono" cannot list pods in the namespace "instavote"

**Either ways, since there are authorization rules set, the user can not make any api calls. Thats when you would create some roles and bind it to the users in the next section.**