THE LINUX FOUNDATION | Training & Certification

Exercise 15.2: Authentication and Authorization

> Kubernetes clusters have two types of users `service accounts` and `normal users`, but `normal users` are assumed to be managed by an outside service. There are no objects to represent them and they cannot be added via an API call, but `service accounts` can be added.
>
> We will use **RBAC** to configure access to actions within a `namespace` for a new contractor, `Developer Dan` who will be working on a new project.

1. Create two namespaces, one for `production` and the other for `development`.

   `student@cp:~$ kubectl create ns development`

   ```
   namespace/development created
   ```

   `student@cp:~$ kubectl create ns production`

   ```
   namespace/production created
   ```

2. View the current clusters and context available. The context allows you to configure the cluster to use, namespace and user for **kubectl** commands in an easy and consistent manner.

   `student@cp:~$ kubectl config get-contexts`

   ```
   CURRENT   NAME                       CLUSTER       AUTHINFO          NAMESPACE
   *         kubernetes-admin@kubernetes   kubernetes   kubernetes-admin
   ```

3. Create a new user `DevDan` and assign a password of `lftr@in`.

   `student@cp:~$ sudo useradd -s /bin/bash DevDan`

   `student@cp:~$ sudo passwd DevDan`

   ```
   Enter new UNIX password:  lftr@in
   Retype new UNIX password:  lftr@in
   passwd: password updated successfully
   ```

4. Generate a private key then Certificate Signing Request (CSR) for `DevDan`. On some Ubuntu 18.04 nodes a missing file may cause an error with random number generation. The **touch** command should ensure one way of success.

   `student@cp:~$ openssl genrsa -out DevDan.key 2048`

   ```
   Generating RSA private key, 2048 bit long modulus
   ......+++
   .........+++
   e is 65537 (0x10001)
   ```

   `student@cp:~$ touch $HOME/.rnd`

   `student@cp:~$ openssl req -new -key DevDan.key \`
   `        -out DevDan.csr -subj "/CN=DevDan/O=development"`

5. Using thew newly created request generate a self-signed certificate using the x509 protocol. Use the CA keys for the Kubernetes cluster and set a 45 day expiration. You'll need to use **sudo** to access to the inbound files.

 LINUX FOUNDATION | Training & Certification

```
student@cp:~$ sudo openssl x509 -req -in DevDan.csr \
     -CA /etc/kubernetes/pki/ca.crt \
     -CAkey /etc/kubernetes/pki/ca.key \
     -CAcreateserial \
     -out DevDan.crt -days 45
```

```
1  Signature ok
2  subject=/CN=DevDan/O=development
3  Getting CA Private Key
```

6. Update the access config file to reference the new key and certificate. Normally we would move them to a safe directory instead of a non-root user's home.

```
student@cp:~$ kubectl config set-credentials DevDan \
     --client-certificate=/home/student/DevDan.crt \
     --client-key=/home/student/DevDan.key
```

```
1  User "DevDan" set.
```

7. View the update to your credentials file. Use **diff** to compare against the copy we made earlier.

```
student@cp:~$ diff cluster-api-config .kube/config
```

```
1  16a,19d15
2  > - name: DevDan
3  >   user:
4  >     as-user-extra: {}
5  >     client-certificate: /home/student/DevDan.crt
6  >     client-key: /home/student/DevDan.key
```

8. We will now create a context. For this we will need the name of the cluster, namespace and CN of the user we set or saw in previous steps.

```
student@cp:~$ kubectl config set-context DevDan-context \
     --cluster=kubernetes \
     --namespace=development \
     --user=DevDan
```

```
1  Context "DevDan-context" created.
```

9. Attempt to view the `Pods` inside the `DevDan-context`. Be aware you will get an error.

```
student@cp:~$ kubectl --context=DevDan-context get pods
```

```
1  Error from server (Forbidden): pods is forbidden: User "DevDan"
2  cannot list pods in the namespace "development"
```

10. Verify the context has been properly set.

```
student@cp:~$ kubectl config get-contexts
```

```
1  CURRENT   NAME                 CLUSTER      AUTHINFO            NAMESPACE
2            DevDan-context       kubernetes   DevDan              development
3  *         kubernetes-admin@kubernetes kubernetes kubernetes-admin
```

11. Again check the recent changes to the cluster access config file.

```
student@cp:~$ diff cluster-api-config .kube/config
```

```
1  <output_omitted>
```

12. We will now create a YAML file to associate RBAC rights to a particular namespace and `Role`.

```
student@cp:~$ vim role-dev.yaml
```

YA ML   **role-dev.yaml**

```
1  kind: Role
2  apiVersion: rbac.authorization.k8s.io/v1
3  metadata:
4    namespace: development
5    name: developer
6  rules:
7  - apiGroups: ["", "extensions", "apps"]
8    resources: ["deployments", "replicasets", "pods"]
9    verbs: ["list", "get", "watch", "create", "update", "patch", "delete"]
10 # You can use ["*"] for all verbs
```

13. Create the object. Check white space and for typos if you encounter errors.

    student@cp:~$ kubectl create -f role-dev.yaml

    ```
    1  role.rbac.authorization.k8s.io/developer created
    ```

14. Now we create a `RoleBinding` to associate the `Role` we just created with a user. Create the object when the file has been created.

    student@cp:~$ vim rolebind.yaml

YA ML   **rolebind.yaml**

```
1  kind: RoleBinding
2  apiVersion: rbac.authorization.k8s.io/v1
3  metadata:
4    name: developer-role-binding
5    namespace: development
6  subjects:
7  - kind: User
8    name: DevDan
9    apiGroup: ""
10 roleRef:
11   kind: Role
12   name: developer
13   apiGroup: ""
```

    student@cp:~$ kubectl create -f rolebind.yaml

    ```
    1  rolebinding.rbac.authorization.k8s.io/developer-role-binding created
    ```

15. Test the context again. This time it should work. There are no `Pods` running so you should get a response of `No resources found`.

    student@cp:~$ kubectl --context=DevDan-context get pods

    ```
    1  No resources found in development namespace.
    ```

16. Create a new pod, verify it exists, then delete it.

    student@cp:~$ kubectl --context=DevDan-context \
          create deployment nginx --image=nginx

    ```
    1  deployment.apps/nginx created
    ```

    student@cp:~$ kubectl --context=DevDan-context get pods

```
1  NAME                      READY     STATUS          RESTARTS    AGE
2  nginx-7c87f569d-7gb9k     1/1       Running         0           5s
```

```
student@cp:~$ kubectl --context=DevDan-context delete \
      deploy nginx
```

```
1  deployment.apps "nginx" deleted
```

17. We will now create a different context for production systems. The `Role` will only have the ability to view, but not create or delete resources. Begin by copying and editing the `Role` and `RoleBindings` YAML files.

```
student@cp:~$ cp role-dev.yaml role-prod.yaml
```

```
student@cp:~$ vim role-prod.yaml
```

role-prod.yaml

```
1  kind: Role
2  apiVersion: rbac.authorization.k8s.io/v1
3  metadata:
4    namespace: production         #<<- This line
5    name: dev-prod                #<<- and this line
6  rules:
7  - apiGroups: ["", "extensions", "apps"]
8    resources: ["deployments", "replicasets", "pods"]
9    verbs: ["get", "list", "watch"] #<<- and this one
```

```
student@cp:~$ cp rolebind.yaml rolebindprod.yaml
```

```
student@cp:~$ vim rolebindprod.yaml
```

rolebindprod.yaml

```
1   kind: RoleBinding
2   apiVersion: rbac.authorization.k8s.io/v1
3   metadata:
4     name: production-role-binding  #<-- Edit to production
5     namespace: production          #<-- Also here
6   subjects:
7   - kind: User
8     name: DevDan
9     apiGroup: ""
10  roleRef:
11    kind: Role
12    name: dev-prod                 #<-- Also this
13    apiGroup: ""
```

18. Create both new objects.

```
student@cp:~$ kubectl create -f role-prod.yaml
```

```
1  role.rbac.authorization.k8s.io/dev-prod created
```

```
student@cp:~$ kubectl create -f rolebindprod.yaml
```

```
1  rolebinding.rbac.authorization.k8s.io/production-role-binding created
```

19. Create the new context for production use.

```
student@cp:~$ kubectl config set-context ProdDan-context \
        --cluster=kubernetes \
        --namespace=production \
        --user=DevDan
```

```
1  Context "ProdDan-context" created.
```

20. Verify that user DevDan can view pods using the new context.

```
student@cp:~$ kubectl --context=ProdDan-context get pods
```

```
1  No resources found in production namespace.
```

21. Try to create a `Pod` in production. The developer should be `Forbidden`.

```
student@cp:~$ kubectl --context=ProdDan-context create \
        deployment nginx --image=nginx
```

```
1  Error from server (Forbidden): deployments.apps is forbidden:
2  User "DevDan" cannot create deployments.apps in the
3  namespace "production"
```

22. View the details of a role.

```
student@cp:~$ kubectl -n production describe role dev-prod
```

```
1  Name:          dev-prod
2  Labels:        <none>
3  Annotations:   kubectl.kubernetes.io/last-applied-configuration=
4  {"apiVersion":"rbac.authorization.k8s.io/v1","kind":"Role"
5  ,"metadata":{"annotations":{},"name":"dev-prod","namespace":
6  "production"},"rules":[{"api...
7  PolicyRule:
8    Resources           Non-Resource URLs   Resource Names   Verbs
9    ---------           -----------------   --------------   -----
10   deployments         []                  []               [get list watch]
11   deployments.apps    []                  []               [get list watch]
12  <output_omitted>
```

23. Experiment with other subcommands in both contexts. They should match those listed in the respective roles.

24. **OPTIONAL CHALLENGE STEP:** Become the `DevDan` user. Solve any missing configuration errors. Try to create a deployment in the `development` and the `production` namespaces. Do the errors look the same? Configure as necessary to only have two contexts available to DevDan.

```
DevDan@cp:~$ kubectl config get-contexts
```

```
1  CURRENT   NAME             CLUSTER      AUTHINFO   NAMESPACE
2  *         DevDan-context   kubernetes   DevDan     development
3            ProdDan-context  kubernetes   DevDan     production
```