Understanding Kubernetes Authentication

58m 52s left

Open Cloud Environment
39%
Ready to login
Average setup time: 2m 26s

Open Code Environment
100%
Setup completed

Credentials

Account ID
137225222 Copy

Username
student Copy

Password
Ca1_Qv3E\ Copy

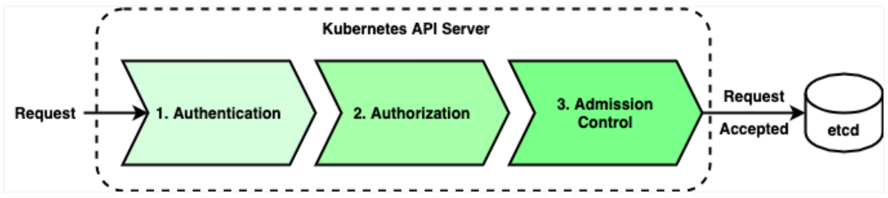
Region
US West 2 Copy

PEM PPK
Download Download

Bridge Connection
Loading...

Introduction

All requests sent to Kubernetes go through the Kubernetes API, which is hosted by the Kubernetes API Server. When you use `kubectl` to create a Deployment, `kubectl` sends a request to the API Server to create it. The Deployment object gets stored in the cluster's data store (etcd) once the request is accepted and the cluster controllers begin to create the Deployment's Pods. However, before the request is accepted, the API Server must allow it through its three layers of access control depicted below:



1. Authentication: Requests sent to the API server are authenticated to prove the identity of the requester, be it a normal user or a service account, and are rejected otherwise.
2. Authorization: The action specified in the request must be in the list of actions the authenticated user is allowed to perform or it is rejected.
3. Admission Control: Authorized requests must then pass through all of the admission controllers configured in the cluster (excluding read-only requests) before any action is performed.

In this lab step, you will explore authentication in the lab's Kubernetes cluster. How does `kubectl` know who is sending the request? How does the cluster recognize the sender? These are the questions that are answered in this lab step.

Instructions

1. Use `kubectl` to get the Pods in the `default` Namespace:

Support

Copy code

Skip to content Press option + Q to open this menu

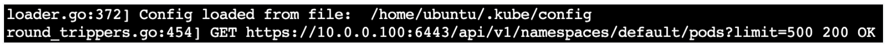
Need help? Contact our support team

was authenticated which is the first layer in the API Server's access control.

2. Re-issue the same command but with log level 6 verbosity (−−v=6):

Copy code

```
kubectl get pods --v=6
```



```
loader.go:372] Config loaded from file:  /home/ubuntu/.kube/config
round_trippers.go:454] GET https://10.0.0.100:6443/api/v1/namespaces/default/pods?limit=500 200 OK
```

The log **Config loaded from file** indicates the configuration file used by `kubectl`. This file is referred to as the kubeconfig file and is in the default location (`.kube/config` in the user's home directory). The following log summarizes the REST API request sent to the API Server. The kubeconfig file includes:

1. Information about the cluster, such as the server address
2. Information about users to authenticate as including certificates

You will see this next.

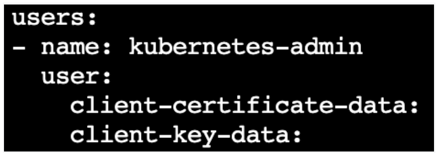
3. Display the contents of the kubeconfig file:

Copy code

```
cat /home/ubuntu/.kube/config
```

The file is in YAML format. Most of the contents are certificate and key data, but observe that there are several top-level keys including **clusters**, **contexts**, **current-context**, and **users**.

The **users** list includes one user (**kubernetes-admin**) and provides a client certificate (**client-certificate-data**) and client key (**client-key-data**) to authenticate requests (certificate and key values omitted in the screenshot):

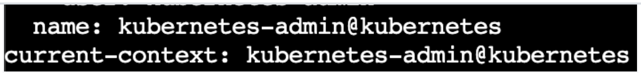


```
users:
- name: kubernetes-admin
  user:
    client-certificate-data:
    client-key-data:
```

Client key and certificate is the default way to authenticate requests in Kubernetes clusters although other authentication modules are supported, including passwords and tokens. A request will pass authenticate if any of the authentication modules successfully authenticates the request.

Skip to content Press option + Q to open this menu

ontext keys:



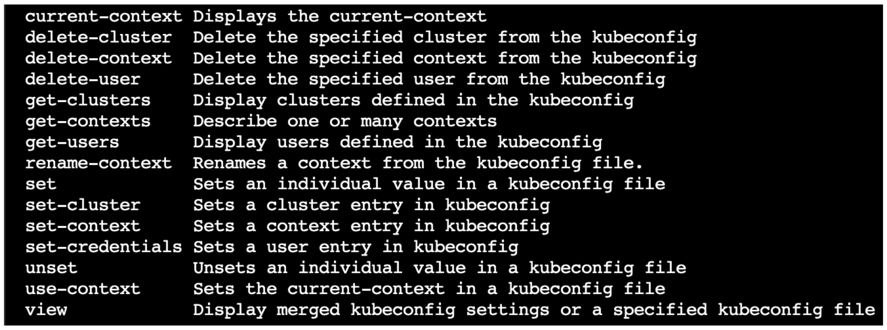
```
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
```

A context is a triple of a **cluster** (**kubernetes**), a **user** (**kubernetes-admin**), and a Namespace (if not specified, the `default` Namespace is used). The context is also given a name for reference (**kubernetes-admin@kubernetes**).
The **current-context** sets the context that will be used by `kubectl` by default. To manage the configuration of `kubectl`, you can use its `config` command.

4. View the `config` commands provided by `kubectl`:

Copy code

```
kubectl config --help
```



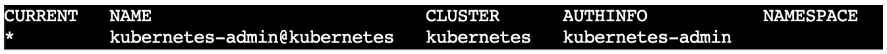
```
  current-context Displays the current-context
  delete-cluster  Delete the specified cluster from the kubeconfig
  delete-context  Delete the specified context from the kubeconfig
  delete-user     Delete the specified user from the kubeconfig
  get-clusters    Display clusters defined in the kubeconfig
  get-contexts    Describe one or many contexts
  get-users       Display users defined in the kubeconfig
  rename-context  Renames a context from the kubeconfig file.
  set             Sets an individual value in a kubeconfig file
  set-cluster     Sets a cluster entry in kubeconfig
  set-context     Sets a context entry in kubeconfig
  set-credentials Sets a user entry in kubeconfig
  unset           Unsets an individual value in a kubeconfig file
  use-context     Sets the current-context in a kubeconfig file
  view            Display merged kubeconfig settings or a specified kubeconfig file
```

These commands can be used to safely write to kubeconfig files using the **delete**, **set**, **unset**, and **use** commands.

5. Enter the following `config` command to get a summary of the contexts available in a kubeconfig file:

Copy code

```
kubectl config get-contexts
```



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

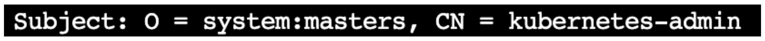
This view is useful for showing you all of the configured contexts and which is the **CURRENT** context. If there is no current context set, there will be no * in the first column. In this lab, there is only one context, but you could have several contexts in practice. There are also multiple contexts for different clusters in the Kuberenetes certification exams. If the current context is not

to change it.

Copy code

```
grep "client-cert" ~/.kube/config | \
  sed 's/\(.*client-certificate-data: \)\(.*\)/\2/' | \
  base64 --decode \
  > cert.pem
openssl x509 -in cert.pem -text -noout
```

For this lab, the line to focus on is as follows:



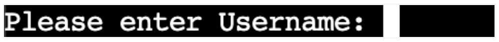
```
Subject: O = system:masters, CN = kubernetes-admin
```

The **Subject** shows the **kubernetes-admin** common name (**CN**) and **system:masters** organization (**O**). Kubernetes maps the common name to users and the organization (if present) to groups. The cluster can verify the certificate is valid and therefore any request using the certificate is authenticated as the **kuberentes-admin** user.

7. Back up the kubeconfig before removing the user and observing the impact on a request:

Copy code

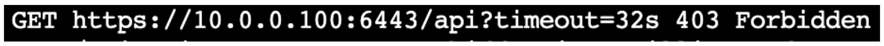
```
cp .kube/config .kube/config.orig
sed -i '15,$d' .kube/config
kubectl get pods --v=6
```



```
Please enter Username:
```

Without a certificate username and password authentication is fallen back to.

8. Enter any username and password at the prompts and observe the request is **Forbidden**:



```
GET https://10.0.0.100:6443/api?timeout=32s 403 Forbidden
```

No users are configured for password authentication in the cluster so any combination of username and password will be forbidden. The same result would happen if a client certificate expired or was invalid for another reason.

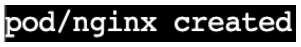
9. Press *ctrl+c* to end the credential prompts and replace the kubeconfig with it's original:

Copy code

10. Create a simple nginx pod:

Copy code

```
kubectl run nginx --image=nginx
```



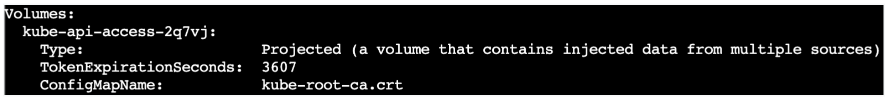
```
pod/nginx created
```

View the projected volume:

Copy code

```
kubectl describe pod nginx
```

At the bottom of the output, under **Volumes**:



```
Volumes:
  kube-api-access-2q7vj:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
```

The projected volume consists of:

- A token acquired from kube-apiserver that will expire after 1 hour by default or when the pod is deleted. The token is bound to the pod and allows communication with the kube-apiserver.
- A ConfigMap containing a CA bundle used for verifying connections to the kube-apiserver.

11. View the value of the token by entering the following:

Copy code

```
kubectl exec nginx -- cat
/var/run/secrets/kubernetes.io/serviceaccount/token && echo
```

12. Delete the pod:

Copy code

```
kubectl delete pod nginx
```

of if they are sent from `kubectl`, using client libraries or REST API requests (in fact `kubectl` and client libraries are also sending REST API requests but abstracting the details away from you). You understood the role of kubeconfig files, contexts, and `config` commands to ensure that `kubectl` is properly configured to communicate with the target cluster. You also saw how ServiceAccounts use tokens for authentication.

Did you like this step?

End Lab Back Next Step

ABOUT US

About Cloud Academy

About QA

About Circus Street

COMMUNITY

Join Discord Channel

HELP

Help Center