pyrtd

GeorgeRaven

May 19, 2021

TABLE OF CONTENTS

CHAPTER

ONE

USER AUTHENTICATION

By default the Kube-api-server does not have user authentication. Instead you can use signed certificates to authenticate a user and the users group, as the kube-api-server will trust certificates signed by its own certificate-authority and has an internal mechanism known as certificate signing requests with which approval can be sought to sign certificates with the internal CA.cert.

1.1 Certificate as User Authentication Step-by-Step

To have a certificate represent a user there are 3 requirements of the certificate:

- the CN (common name field) is populated with the username desired, or more specifically the username the administrator has/ expects for you with your roles/ role bindings.
- the O (organisation filed) is populated with the groups of the user that the administrator is expecting (can specify multiple).
- the certificate is signed by the kubernetes certificate authority (part of any kubernetes cluster)

To do this you must have completed the following *Prerequisites* -> *Generate the Users Private Key* -> *Generate a Certificate Signing Request* -> *Use CSR in Kubernetes yaml File to Request Signing by Administrator*.

1.1.1 Prerequisites

- OpenSSL
  - Windows
  - linux
  - osx

1.1.2 Generate the Users Private Key

First to get this certificate to represent a user we need to generate a private key. This private key is a secret that only the user should have as for all intensive purposes it is the user when embedded in a certificate as far as the kube-api is concerned.

Bash shell example; Generating private key

```
openssl genrsa -out username.key 2048
```

Replace `username` with the desired `username`

1.1.3 Generate a Certificate Signing Request

Warning: The `username` and `group` (s) used here must match what the roles/ role bindings are expecting, or else they simply wont allow the user under RBAC to access anything even if they are accepted.

Now that we have the private key we want to use, we need to create a certificate signing request (CSR) to associate the private key with a certificate that contains all the additional information we want it to be related to like the common name, and organisation fields.

Bash shell example; certificate signing request generation

```
openssl req -new -key username.key -out username.csr -subj "/CN=username/O=group"
```

OR

Bash shell example; multi group certificate signing request generation

```
openssl req -new -key username.key -out username.csr -subj "/CN=username/O=group1/
↪O=group2"
```

Warning: Make sure you replace `username` and `group` with the correct/ desired `username` and `group` specified by your cluster administrator. (the last `username` and `group` are not highlighted as quoted)

Note: It is advisable to check your now created CSR which should if you followed the above be called `username.csr`. You can check your CSR by:

Bash shell example; inspect CSR

```
openssl req --noout -text -in username.csr
```

You should see something akin to the following towards the top:

```
Certificate Request:
  Data:
      Version: 1 (0x0)
      Subject: CN = username, O = group
      Subject Public Key Info:
          Public Key Algorithm: rsaEncryption
              RSA Public-Key: (2048 bit)
              Modulus:
```

If these are not correct, create a new certificate now, otherwise you will be waiting for acceptance to only get rejected because they are incorrect later.

See also:

- certificate signing request generation

1.1.4 Use CSR in Kubernetes yaml File to Request Signing by Administrator

Now that we have a private-key, and a certificate signing request (.csr) file, we can now ask the kubernetes api to consider us for authentication, where an admin can accept/ decline us or a pre-defined rule can accept/ decline us.

To do so we need to define a short YAML file as seen below:

```yaml
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: <YOURNAME>
spec:
  groups:
  - system:authenticated
  request: <YOUR certificate_signing_request.csr GOES HERE as BASE64 ENCODED>
  signerName: kubernetes.io/kube-apiserver-client
  usages:
  - client auth
```

We need to substitute our name or more specifically the name we used earlier in the CN field. If you have forgotten what it was or need to double check please refer to *Generate a Certificate Signing Request* which tells you how to inspect the certificate as well as generate it. We also need to insert our certificate in one line and Base64 encoded. If encoding the certificate gives you any problems try our Helm helper template (you will need to have Helm installed) Located here. Simply download the files pointed to here and run:

Bash shell example; Helm CSR template generator

```
helm template folder/path/to/csr --set username=username,csr=secrets/file-name.csr
```

This will paste the correct contents to your terminal which you can paste into a file, should it be correct. There should be no blank entries in particular where there are comments, there should be text between the comment and the preceeding colon of the same line I.E `request:  jhaiusdf912iudifgakdujh27dhf # comment` NOT `request:  # comment`. check both request and name. If one or the other is blank that means you have set the names or paths wrong.

where:

- `folder/path/to/` is the path to the *folder* containing the Chart.yaml file of the helm chart.
- `secrets/file-name` is the relative path from the above *folder* to find your csr file created in *Generate a Certificate Signing Request*.

Note:

our CSR should be embedded in the above file next to request. This CSR should be embedded as Base64. If you are on linux you can use the following to get the properly formatted base64 string to insert:

```
cat username.csr | base64 | tr -d "n"
```

Now that we have a CSR yaml file lets call it `csr`.yaml for consistency. We can submit this to kubernetes and await approval. We can do this in two ways:

- Ask our administrator (or someone else with permissions) to submit the file to the api-server for approval
- Submit the file ourselves for approval to cut out the middleman, if we have credentials to do so, like some generic CSR submitting user (Ask your admin if there is one).

Please do mention the CSR submission to someone who can approve it, as an hour after submission it will automatically be removed from the kube-api-server by default, effectively being declined.

Note: An authorised kubernetes administrator will then likeley run:

Bash shell example; get/ check csrs'

```
kubectl get csr -o wide
```

Bash shell example; approve or decline the csr

```
kubectl certificate approve|decline username
```

Bash shell example; get CSR if approved, and give to user without perms

```
kubectl get csr username -o jsonpath='{.status.certificate}'| base64 -d > username.crt
```

1.1.5 Use Approved CSR to Authenticate to Kube-Api

We now should have both a private key (`username`.key) from *Generate the Users Private Key*, and a signed certificate (`username`.crt) from kubernetes in *Use CSR in Kubernetes yaml File to Request Signing by Administrator*. The last thing we need is the public certificate-authority certificate so that our client when connecting to the kube-api can also verify that it is also signed by the certificate-authority, to prevent redirecting attacks of some sort. You will either be given the certificate-authority certificate (usually called ca.cert) by your admin, if not you should request it as you will need it to communicate to the kube-api server to issue your commands.

Given these 3 things along with we know our `username` and more than likely know the ip address where the kube-api is running we can finally configure out kubectl config file.

I have provided another Helm template so you can generate your kubectl config with it or you can replicate a similar example to the documentation / use kubectl commands to generate the config. This template is at https://github.com/DreamingRaven/deep-kube-docs/tree/master/helpers/kubectl

simply run as before:

Bash shell example; generate a kubectl config

```
helm template folder/path/to/kubectl --set username=username,ca_cert=certificates/ca.
↪cert,user_cert=certificates/username.cert,user_key=certificates/username.key,kube_api_
↪address=https://192.168.1.102:6443
```

where:

- `folder/path/to/` is the path to the parent folder that contains Chart.yaml exists on your system after you downloaded this helm chart from https://github.com/DreamingRaven/deep-kube-docs/tree/master/helpers/kubectl
- `username` is the username you used throught this process
- `192.168.1.102` is the accessible address of the machine hosting the kube-api-server
- `6443` is the port the kube-api-server is listening on, by default: 6443

See also:

- kubectl config files

CHAPTER

TWO

REMOTE VPN CONNECTIONS WITH LINUX

This section will guide you through connecting to the university VPN using AnyConnect on Linux. Windows users do not need to follow these instructions, but Mac users might.

2.1 Downloading Certificates

In order to view the required certificates (on Firefox):

1. Open the university's WebVPN service in your browser.
2. Click the padlock icon to the left of the URL.
3. Click the arrow to the right of "Connection Secure".
4. Click "More Information".
5. On the Security tab, click "View Certificate".

There should be three tabs at the top. We only need to concern ourselves with the two QuoVadis ones. For each, scroll down to a section called "Miscellaneous", and next to "Download", click "PEM (cert)". This will save each certificate to your system. The filename and save location don't matter.

2.2 Installing Certificates

You then need to open your terminal and `cd` to where you saved your PEM files. For each PEM file, run the following command:

```
sudo openssl x509 -in <CERTNAME>.pem -inform PEM -out <CERTNAME>.crt
```

Here `<CERTNAME>` should be replaced with the name of the certificate. Again, the filenames do not matter, and you can create the CRT files with different names to the PEM files if you wish.

Next, make a new directory for your certificates and copy the CRT files into it (make sure you're only copying the two CRT files we just created to this directory):

```
sudo mkdir /usr/share/ca-certificates/extra
sudo cp *.crt /usr/share/ca-certificates/extra
```

Now we can install the certificates:

```
sudo dpkg-reconfigure ca-certificates
```

On the screen that appears, select "ask". You should then be prompted with a list of certificates. The ones you copied over will not have asterisks next to them, so you need to navigate to them using the arrow keys, and press SPACE to select it. After you've selected them, press TAB, then RETURN to confirm your choice. If you see `2 added, 0 removed; done.` anywhere in the subsequent output, it was successful.

You can now go to AnyConnect and connect as you would normally! It might take a few tries for it to work (normally 2, sometimes 3). If you successfully connect, but are then disconnected, try again – it should work the second time. If you simply cannot connect at all, you may need to consult the IT department.

2.3 Testing your permissions

If you are a professor, PhD student, or network administrator, you should be good to go. If you are an undergraduate student, you need to have an ad-hoc account created for you. You may also need to do this if you are a Masters student.

To test whether you have proper access run the following command while connected to the VPN:

```
ping ausers03
```

If the output reads `Temporary failure in name resolution`, you are not connected. Make sure you connected successfully before. If the output reads `Destination port unreachable`, you need an ad-hoc account; see the section below. If the output seems normal, you should be good to go!

2.3.1 Creating an ad-hoc account

An ad-hoc account is one created for a student and verified by a professor in order to provide elevated access to the remote internals. You should contact the IT department, and forward the response to your project supervisor (if you are currently doing a dissertation) or your personal tutor. You should then have the correct permissions, but will need to connect to the VPN using this account rather than your student one.

CHAPTER

THREE

USAGE

This section will briefly introduce you to how you can use a kubernetes cluster for some basic tasks.

3.1 prerequisites

- You have a user with authentication to the cluster and have created your kubectl config from *Use Approved CSR to Authenticate to Kube-Api*.
- You have kubectl installed and accessible.
- You have a cluster to connect to.

3.2 Testing your kube-config

Before worrying about making your generated config file permenant you can test it works by making a simple request with it by using the –kubeconfig flag like:

Bash shell example; test your kubectl config

```
kubectl --kubeconfig path/to/kube/config get nodes
```

Where you should see an output like:

```
NAME     STATUS   ROLES                  AGE   VERSION
host    Ready    control-plane,master   28d   v1.20.2
```

If you get this output you are all set to continue if not you will need to check:

- that the kube-api server is actually alive and listening
- that the kube-api server is accessible from your machine
- that you are connecting to the correct kube-api server which you have your certificate signed by previously
- that your kubectl config is correct/ free of errors, and actually contains the base 64 encoded strings of your certificates or if not atleast paths to where those certificates can be found
- that you have the correct permissions to access what you are trying to access e.g the correct namespace

If you do get a “Forbidden” message try using the following instead to check your permissions:

Bash shell example; test your kubectl config

```
kubectl --kubeconfig path/to/kube/config auth can-i get pods --as  USERNAME --namespace␣
↪NAMESPACENAME
```

This command will either tell you yes, or no. If it fails in giving you a yes or no then you aren't authenticating to the server at all, please check roles/ rolebindings/ certificates/ the kubectl config itself.

where:

- username is just what username the server expects I.E whatever it signed in the certificates, and has a role/ role-binding for
- namespacename is just what namespace you want to enact these commands in, for example you may only have permissions in a specific namespace. Try asking your administrator if you do not know what this is, but try using either your name or your teams name is a good place to start.

3.3 Making the Kubeconfig Permanent

If this all works consider making your kubeconfig permanent, by putting it where kubectl looks for it by default on your OS. For example on Linux the default kubeconfig location is `${HOME}/.kube/config` if it is there then you have no need to keep issuing the –kubeconfig `path/to/kube/config` option each time you call kubectl.

see: https://kubernetes.io/docs/concepts/configuration/organize-cluster-access-kubeconfig/

CHAPTER

FOUR

NVIDIA-GPU

4.1 Cluster Configuration

Note: Clearly since this is cluster configuration this only applies to administrators. Normal users will have no need to follow this cluster configuration step.

To enable nvidia-container-toolking previously nvidia-docker, by editing /etc/docker/daemon.json since kubernetes does not support the docker –gpu option:

https://docs.nvidia.com/datacenter/cloud-native/kubernetes/dcgme2e.html#install-nvidia-container-toolkit-previously-nvidia-docker2

```
{
   "default-runtime": "nvidia",
   "runtimes": {
        "nvidia": {
            "path": "/usr/bin/nvidia-container-runtime",
            "runtimeArgs": []
      }
   }
}
```

Then we need to install the nvidia operator:

```
helm repo add nvdp https://nvidia.github.io/k8s-device-plugin \
 && helm repo update \
 && helm install --generate-name nvdp/nvidia-device-plugin
```

4.2 Pod Specification

4.3 Test Pods

Test pod for cuda functionality

Manifest example

```
1 apiVersion: v1
2 kind: Pod
```

(continues on next page)

(continued from previous page)

```
metadata:
  name: gpu-operator-test
spec:
  restartPolicy: OnFailure
  containers:
  - name: cuda-vector-add
    image: "nvidia/samples:vectoradd-cuda10.2"
    resources:
      limits:
         nvidia.com/gpu: 1
```

Test job for nvidia-smi

Manifest example

```
apiVersion: batch/v1
kind: Job
metadata:
  name: smi
spec:
  template:
    spec:
      containers:
      - name: smi
        image: docker.io/nvidia/cuda:11.0-base
        command: ['nvidia-smi']
      restartPolicy: OnFailure
```

These docs are to give users/ developers, and admins a brief guide to setting up, testing, and using a Kubernetes cluster. These docs will rely heavily on external, and much more thoroughly explain documentation in places for more complex topics that I could otherwise not hope to explain here so simply.