Exercise 15.1: Working with TLS

Overview

We have learned that the flow of access to a cluster begins with TLS connectivity, then authentication followed by authorization, finally an admission control plug-in allows advanced features prior to the request being fulfilled. The use of `Initializers` allows the flexibility of a shell-script to dynamically modify the request. As security is an important, ongoing concern, there may be multiple configurations used depending on the needs of the cluster.

Every process making API requests to the cluster must authenticate or be treated as an anonymous user.

While one can have multiple cluster root Certificate Authorities (CA) by default each cluster uses their own, intended for intra-cluster communication. The CA certificate bundle is distributed to each node and as a secret to default service accounts. The **kubelet** is a local agent which ensures local containers are running and healthy.

1. View the **kubelet** on both the cp and secondary nodes. The **kube-apiserver** also shows security information such as certificates and authorization mode. As **kubelet** is a **systemd** service we will start looking at that output.

```
student@cp:~$ systemctl status kubelet.service
```

```
    kubelet.service - kubelet: The Kubernetes Node Agent
     Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor preset: en
    Drop-In: /etc/systemd/system/kubelet.service.d
             |__10-kubeadm.conf
  <output_omitted>
```

2. Look at the status output. Follow the CGroup and kubelet information, which is a long line where configuration settings are drawn from, to find where the configuration file can be found.

```
  CGroup: /system.slice/kubelet.service
   |--19523 /usr/bin/kubelet .... --config=/var/lib/kubelet/config.yaml ..
```

3. Take a look at the settings in the `/var/lib/kubelet/config.yaml` file. Among other information we can see the `/etc/kubernetes/pki/` directory is used for accessing the **kube-apiserver**. Near the end of the output it also sets the directory to find other pod spec files.

```
student@cp:~$ sudo less /var/lib/kubelet/config.yaml
```

YAML **config.yaml**

```
1 <output_omitted>
2 rotateCertificates: true
3 runtimeRequestTimeout: 0s
4 shutdownGracePeriod: 0s
5 shutdownGracePeriodCriticalPods: 0s
6 staticPodPath: /etc/kubernetes/manifests
7 streamingConnectionIdleTimeout: 0s
8 syncFrequency: 0s
9 volumeStatsAggPeriod: 0s
```

4. Other agents on the cp node interact with the **kube-apiserver**. View the configuration files where these settings are made. This was set in the previous YAML file. Look at one of the files for cert information.

```
student@cp:~$ sudo ls /etc/kubernetes/manifests/
```

```
etcd.yaml            kube-controller-manager.yaml
kube-apiserver.yaml  kube-scheduler.yaml
```

```
student@cp:~$ sudo less /etc/kubernetes/manifests/kube-controller-manager.yaml
```

```
<output_omitted>
```

5. The use of `tokens` has become central to authorizing component communication. The tokens are kept as **secrets**. Take a look at the current secrets in the `kube-system` namespace.

```
student@cp:~$  kubectl -n kube-system get secrets
```

```
NAME                                 TYPE
 DATA      AGE
bootstrap-token-i3r13t               bootstrap.kubernetes.io/token
 7         5d
<output_omitted>
```

6. Take a closer look at one of the secrets and the token within. The `bootstrap-token` could be one to look at. The use of the Tab key can help with long names. Long lines have been truncated in the output below.

```
student@cp:~$ kubectl -n kube-system get secrets bootstrap-token<Tab> -o yaml
```

```yaml
apiVersion: v1
data:
  auth-extra-groups: c3lzdGVtOmJvb3RzdHJhcHBlcnM6a3ViZWFkbTpkZWZhdWx0LW5vZGUtdG9rZW4=
  expiration: MjAyMy0wNi0wMlQxMjowOTowMlo=
  token-id: NXBvMGo3
  token-secret: MXhzMDgxOG1rcTFyeDQxbg==
  usage-bootstrap-authentication: dHJ1ZQ==
  usage-bootstrap-signing: dHJ1ZQ==
kind: Secret
metadata:
  creationTimestamp: "2023-06-01T12:09:02Z"
  name: bootstrap-token-5po0j7
  namespace: kube-system
  resourceVersion: "209"
  uid: 98219199-4876-4cfa-a4be-586cda27cc6b
type: bootstrap.kubernetes.io/token
```

7. The **kubectl config** command can also be used to view and update parameters. When making updates this could avoid a typo removing access to the cluster. View the current configuration settings. The keys and certs are redacted from the output automatically.

```
student@cp:~$ kubectl config view
```

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: REDACTED
<output_omitted>
```

8. View the options, such as setting a password for the admin instead of a key. Read through the examples and options.

```
student@cp:~$ kubectl config set-credentials -h
```

```
Sets a user entry in kubeconfig
<output_omitted>
```

9. Make a copy of your access configuration file. Later steps will update this file and we can view the differences.

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

10. Explore working with cluster and security configurations both using **kubectl** and **kubeadm**. Among other values, find the name of your cluster. You will need to become `root` to work with **kubeadm**.

```
student@cp:~$ kubectl config <Tab><Tab>
```

```
current-context  get-contexts     set-context      view
delete-cluster   rename-context   set-credentials
delete-context   set              unset
get-clusters     set-cluster      use-context
```

```
student@cp:~$ sudo kubeadm token -h
```

```
<output_omitted>
```

```
student@cp:~$ sudo kubeadm config -h
```

```
<output_omitted>
```

11. Review the cluster default configuration settings. There may be some interesting tidbits to the security and infrastructure of the cluster.

```
student@cp:~$ sudo kubeadm config print init-defaults
```

```
apiVersion: kubeadm.k8s.io/v1beta2
bootstrapTokens:
- groups:
  - system:bootstrappers:kubeadm:default-node-token
  token: abcdef.0123456789abcdef
  ttl: 24h0m0s
  usages:
<output_omitted>
```