Exercise 5.1: Configuring TLS Access

Overview

Using the Kubernetes API, **kubectl** makes API calls for you. With the appropriate TLS keys you could run **curl** as well use a **golang** client. Calls to the `kube-apiserver` get or set a PodSpec, or desired state. If the request represents a new state the **Kubernetes Control Plane** will update the cluster until the current state matches the specified state. Some end states may require multiple requests. For example, to delete a `ReplicaSet`, you would first set the number of replicas to zero, then delete the `ReplicaSet`.

An API request must pass information as JSON. **kubectl** converts `.yaml` to JSON when making an API request on your behalf. The API request has many settings, but must include `apiVersion`, `kind` and `metadata`, and `spec` settings to declare what kind of container to deploy. The `spec` fields depend on the object being created.

We will begin by configuring remote access to the `kube-apiserver` then explore more of the API.

1. Begin by reviewing the **kubectl** configuration file. We will use the three certificates and the API server address.

```
student@cp:~$ less $HOME/.kube/config
```

```
<output_omitted>
```

2. We will create a variables using certificate information. You may want to double-check each parameter as you set it. Begin with setting the `client-certificate-data` key.

```
student@cp:~$ export client=$(grep client-cert $HOME/.kube/config |cut -d" " -f 6)

student@cp:~$ echo $client
```

```
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUM4akNDQWRxZ0F3SUJ
BZ0lJRy9wbC9rWEpNdmd3RFFZSktvWklodmNOQVFFTEJRQXdGVEVUTUJFR0
ExVUUKQXhNS2EzVmlaWEp1WlhSbGN6QWVGdzB4TnpFeU1UTXhOelEyTXpKY
UZ3MHhPREV5TVRNeE56UTJNelJhTURReApGekFWQmdOVkJBb1REbk41YzNS
<output_omitted>
```

3. Almost the same command, but this time collect the `client-key-data` as the `key` variable.

```
student@cp:~$ export key=$(grep client-key-data $HOME/.kube/config |cut -d " " -f 6)

student@cp:~$ echo $key
```

```
<output_omitted>
```

4. Finally set the `auth` variable with the `certificate-authority-data` key.

```
student@cp:~$ export auth=$(grep certificate-authority-data $HOME/.kube/config |cut -d " " -f 6)

student@cp:~$ echo $auth
```

```
<output_omitted>
```

5. Now encode the keys for use with **curl**.

```
student@cp:~$ echo $client | base64 -d - > ./client.pem

student@cp:~$ echo $key | base64 -d - > ./client-key.pem

student@cp:~$ echo $auth | base64 -d - > ./ca.pem
```

6. Pull the API server URL from the config file. Your hostname or IP address may be different.

```
student@cp:~$ kubectl config view |grep server
    server: https://k8scp:6443
```

7. Use **curl** command and the encoded keys to connect to the API server. Use `your` hostname, or IP, found in the previous command, which may be different than the example below.

```
student@cp:~$ curl --cert ./client.pem \
    --key ./client-key.pem \
    --cacert ./ca.pem \
    https://k8scp:6443/api/v1/pods

{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "selfLink": "/api/v1/pods",
    "resourceVersion": "239414"
  },
<output_omitted>
```

8. If the previous command was successful, create a JSON file to create a new pod. Remember to use **find** and search for this file in the tarball output, it can save you some typing.

```
student@cp:~$ vim curlpod.json

{
    "kind": "Pod",
    "apiVersion": "v1",
    "metadata":{
        "name": "curlpod",
        "namespace": "default",
        "labels": {
            "name": "examplepod"
        }
    },
    "spec": {
        "containers": [{
            "name": "nginx",
            "image": "nginx",
            "ports": [{"containerPort": 80}]
        }]
    }
}
```

9. The previous **curl** command can be used to build a `XPOST` API call. There will be a lot of output, including the scheduler and taints involved. Read through the output. In the last few lines the phase will probably show `Pending`, as it's near the beginning of the creation process.

```
student@cp:~$ curl --cert ./client.pem \
   --key ./client-key.pem --cacert ./ca.pem \
   https://k8scp:6443/api/v1/namespaces/default/pods \
   -XPOST -H'Content-Type: application/json' \
   -d@curlpod.json
```

```
{
  "kind": "Pod",
  "apiVersion": "v1",
  "metadata": {
    "name": "curlpod",
<output_omitted>
```

10. Verify the new pod exists and shows a `Running` status.

```
student@cp:~$ kubectl get pods
```

```
NAME      READY   STATUS    RESTARTS   AGE
curlpod   1/1     Running   0          45s
```