Importing external or additional TLS certificates to Che

Contents

Network communications between the components of a Che installation and the started workspaces, are all secured through the TLS protocol, and thus require the use of trusted certificate authorities.

In some cases it can be necessary to add TLS certificates to the Che installation, so that every Che component will consider them as signed by a trusted CA.

Typical cases that may require this addition are:

- when the underlying Kubernetes cluster uses TLS certificates signed by a CA that is not trusted,
- when Che server or workspace components connect to external services such as Keycloak or a Git server that use TLS certificates signed by an untrusted CA.

To store those certificates, Che uses a dedicated ConfigMap. Its default name is `ca-certs` but Che allows configuring its name. On OpenShift, the Che operator even injects cluster trusted certificates into this ConfigMap automatically.

At Che installation time

Prerequisites

- The `kubectl` tool is available.
- You are ready to create `CheCluster` custom resource.

Procedure

1. Save the certificates you need to import, to a local file system.

CAUTION

- Certificate files are typically stored as Base64 ASCII files, such as `.pem`, `.crt`, `.ca-bundle`. But, they can also be binary-encoded, for example, as `.cer` files. All Secrets that hold certificate files should use the Base64 ASCII certificate rather than the binary-encoded certificate.
- Che already uses some reserved file names to automatically inject certificates into the ConfigMap, so you should avoid using the following reserved file names to save your certificates:
  - `ca-bundle.crt`
  - `ca.crt`

2. Create a new ConfigMap with the required TLS certificates:

```
$ kubectl create configmap ca-certs --from-file=<certificate-file-path> -n=<che-namespace-name>
```

   To apply more than one certificate, add another `--from-file=<certificate-file-path>` option to the above command.

3. During the installation process, when creating the `CheCluster` custom resource, take care of configuring the right name for the created ConfigMap.

   For a Che Operator deployment, ensure you add the `spec.server.ServerTrustStoreConfigMapName` field with the name of the ConfigMap, to the `CheCluster` Custom Resource you will create during the installation:

```yaml
spec:
  server:
    ...
    spec.server.ServerTrustStoreConfigMapName: ca-certs
```

For a Che Helm Chart deployment, ensure you override the `global.tls.serverTrustStoreConfigMapName` Helm Chart property with the name of the ConfigMap when installing the Che Helm Chart. For this you should add the following arguments to the Helm command line:

```
--set global.tls.serverTrustStoreConfigMapName=ca-certs
```

On already-running Che installations

Prerequisites

- The `kubectl` tool is available.
- You should first define the name of the ConfigMap you will use to import certificates:

On instances of Che deployed with the Che Operator, retrieve the name of the ConfigMap by reading the `spec.server.ServerTrustStoreConfigMapName` `CheCluster` Custom Resource property:

```
$ get checluster eclipse-che -n <che-namespace-name> -o json-path --jsonpath=
{.spec.server.serverTrustStoreConfigMapName}
```

On instances of Che deployed with the Che Helm Chart deployment, retrieve the name of the ConfigMap by reading the `global.tls.serverTrustStoreConfigMapName` property from the Helm Chart:

```
$ helm get values che --all --output json | jq -r '.global.tls.serverTrustStoreConfigMapName'
```

NOTE

If the existing installation did not define any name for the ConfigMap, just use `ca-certs` .

Procedure

1. Save the certificates you need to import, to a local file system.

CAUTION

- Certificate files are typically stored as Base64 ASCII files, such as `.pem`, `.crt`, `.ca-bundle`. But, they can also be binary-encoded, for example, as `.cer` files. All Secrets that hold certificate files should use the Base64 ASCII certificate rather than the binary-encoded certificate.
- Che already uses some reserved file names to automatically inject certificates into the ConfigMap, so you should avoid using the following reserved file names to save your certificates:
  - `ca-bundle.crt`
  - `ca.crt`

2. Add the required TLS certificates in the ConfigMap:

```
$ kubectl create configmap <config-map-name> --from-file=<certificate-file-path> -n=<che-namespace-name> -o yaml --
dry-run | kubectl apply -f -
```

To apply more than one certificate, add another `--from-file=<certificate-file-path>` option to the above command.

3. Configure the Che installation to use the ConfigMap:

For a Che Operators deployment:

1. Edit the `spec.server.ServerTrustStoreConfigMapName` `CheCluster` Custom Resource property to match the name of the ConfigMap:

```
$ kubectl patch checluster eclipse-che -n <che-namespace-name> --type=json -p '[{"op": "replace", "path":
"/spec/server/serverTrustStoreConfigMapName", "value": "<config-map-name>"}]'
```

For a Che Helm Chart deployment:

1. Clone the che project.
2. Go to the `deploy/kubernetes/helm/che` directory.
3. Update the name of the configMap Che will use, by editing the `global.tls.serverTrustStoreConfigMapName` Helm Chart property to match the created or updated ConfigMap:

```
$ helm upgrade che -n che --set global.tls.serverTrustStoreConfigMapName=<config-map-name> \
   --set global.ingressDomain=<kubernetes-cluster-domain> .
```

When using Minikube to run Che, substitute *<kubernetes-cluster-domain>* with `$(minikube ip).nip.io`.

4. Restart the Che operator, the Che server and Keycloak to load the new certificates:

```
$ kubectl rollout restart -n <che-namespace-name> deployment/che-operator
$ kubectl rollout restart -n <che-namespace-name> deployment/che/keycloak
$ kubectl rollout restart -n <che-namespace-name> deployment/che
```

Verification

If you added the certificates without error, the Che server starts and obtains Keycloak configuration over HTTPS. Otherwise here is a list of things to verify:

- In case of a Che Operator deployment, the `CheCluster` attribute `serverTrustStoreConfigMapName` value matches the name of the ConfigMap. Get the value using the following command :

```
$ kubectl get -o json checluster/eclipse-che -n <che-namespace-name> | jq .spec.server.serverTrustStoreConfigMapName
```

- Che Pod Volumes list contains one Volume that uses the ConfigMap as data-source. To get the list of Volumes of the Che Pod:

```
$ kubectl get pod -o json <che-pod-name> -n <che-namespace-name> | jq .spec.volumes
```

- Che mounts certificates in folder `/public-certs/` of the Che server container. This command returns the list of files in that folder:

```
$ kubectl exec -t <che-pod-name> -n <che-namespace-name> -- ls /public-certs/
```

- In the Che server logs there is a line for every certificate added to the Java truststore, including configured Che certificates.

```
$ kubectl logs <che-pod-name> -n <che-namespace-name>
(...)
Found a custom cert. Adding it to java trust store based on /usr/lib/jvm/java-1.8.0/jre/lib/security/cacerts
(...)
```

- $Che server Java trustore contains the certificates. The certificates SHA1 fingerpints are among the list of the SHA1 of the certificates included in the trustore returned by the following command:

```
$ kubectl exec -t <che-pod-name> -n che -- keytool -list -keystore /home/che/cacerts
Your keystore contains 141 entries

(...)
```

  To get the SHA1 hash of a certificate on the local filesystem:

```
$ openssl x509 -in <certificate-file-path> -fingerprint -noout
SHA1 Fingerprint=3F:DA:BF:E7:A7:A7:90:62:CA:CF:C7:55:0E:1D:7D:05:16:7D:45:60
```

Contents