IBM Cloud Private 3.1.2

**Lab Exercise # Security 1**

**Enable TLS with custom CA Issuer and certificate at Ingress**

**Duration: 45 mins**

**Objective**

The objective of this lab is to know how to enable TLS communication in the applications. Also get to know different ways of using certificates in IBM Cloud Private environment and manage the Issuers.

**Pre-requisites**

We are going to use a helloworld application for this lab.

Deploy a helloworld app – Save this as helloworld.yaml file and deploy the application

***kubectl apply -f*** helloworld.yaml-n <namespace>

apiVersion: v1

kind: Service

metadata:

name: helloworld

labels:

app: helloworld

spec:

ports:

- port: 5000

name: http

selector:

app: helloworld

---

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: helloworld-v1

spec:

replicas: 1

template:

metadata:

labels:

app: helloworld

version: v1

spec:

containers:

- name: helloworld

image: istio/examples-helloworld-v1

resources:

requests:

cpu: "100m"

imagePullPolicy: IfNotPresent #Always

ports:

- containerPort: 5000

**Instructions**

1. Check the Issuer in the IBM Cloud Private environment.

$ kubectl get clusterissuer

NAME AGE

icp-ca-issuer 132d

Get the issuer specific to the namespace, if any available

$ kubectl get issuer

No resources found.

~~NAME AGE~~

~~hello-deployment-tls 1h~~

~~hello-self-tls 1h~~

In the next step, Create a custom Issuer specific to namespace and use that in Ingress to access a hello world application

1. Create a self-signed Issuer. Use the following hello-myself-tls.yaml file to define a self-signed Issuer. Replace <namespace> with your namespace.

apiVersion: certmanager.k8s.io/v1alpha1

kind: Issuer

metadata:

name: hello-myself-tls

namespace: **<namespace>**

spec:

selfSigned: {}

kubectl apply -f hello-myself-tls.yaml

1. After you create the self-signed Issuer, create a CA certificate that references the self-signed Issuer and specifies the isCA field. Replace the namespace <namespace> with your namespace.

apiVersion: certmanager.k8s.io/v1alpha1

kind: Certificate

metadata:

name: hello-cert-tls

namespace: **<namespace>**

spec:

# name of the tls secret to store

# the generated certificate/key pair

secretName: **hello-deployment-tls-ca-key-pair**

isCA: true

issuerRef:

# issuer created in step 1

name: hello-myself-tls

kind: Issuer

commonName: "foo.bar"

dnsNames:

# one or more fully-qualified domain name

# can be defined here

- foo.bar

kubectl apply -f hello-cert-tls.yaml

1. Edit the following sample of an Issuer that references the previous secret. Edit the <name> and <namespace> from the metadata section of the **self-ca-issuer**.yaml file. Be sure that secretName from the spec section matches the secretName from the previous step:

apiVersion: certmanager.k8s.io/v1alpha1

kind: Issuer

metadata:

name: **self-ca-issuer**

namespace: blueworld

spec:

ca:

secretName: **hello-deployment-tls-ca-key-pair**

kubectl apply -f self-ca-issuer.yaml

**Now a custom issuer called ‘self-ca-issuer’ is created, you can create custom CA issuer with third party or enterprise provider.**

1. Define the certificate using the custom CA issuer created in Step 3. helloworld.x.x.x.x.nip.io is the CN for the certificate. X.X.X.X is the IP address of proxy.

Replace <namespace> with your namespace

helloworld-tls-certs.yaml

apiVersion: certmanager.k8s.io/v1alpha1

kind: Certificate

metadata:

name: helloworld-tls-certs

namespace: <namespace>

spec:

# name of the tls secret to store

# the generated certificate/key pair

secretName: **helloworld-tls-certs**

isCA: true

issuerRef:

# issuer created in step 1

name: **self-ca-issuer**

kind: Issuer

commonName: helloworld.x.x.x.x.nip.io

dnsNames:

# one or more fully-qualified domain name

# can be defined here

- helloworld.x.x.x.x.nip.io

1. Add the Secret to the Kubernetes Ingress. The following step defines a TLS-enabled Kubernetes Ingress that is integrated with cert-manager. Here, **helloworld-tls-certs** matches the secretName that you previously defined and host matches the DNS name that you previously defined in the certificate.

helloworld-ingress-tls.yaml

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: helloworld-ingress-tls

annotations:

kubernetes.io/ingress.class: "nginx"

ingress.kubernetes.io/rewrite-target: "/"

spec:

tls:

# k8s ingress defines different tls certificates

# for each nginx server blocks.

# k8s ingress default cert is used if

# no host-specific secret specified

- hosts:

# this is the fully-qualified domain name

# of the first server block

- hello.9.202.178.184.nip.io

# certificate hello-k8s-ingress-tls-1

# is only used by foo1.bar1

secretName: **helloworld-tls-certs**

rules:

# each server block redirects request

# to its own backend service

- host: helloworld.x.x.x.x.nip.io

http:

paths:

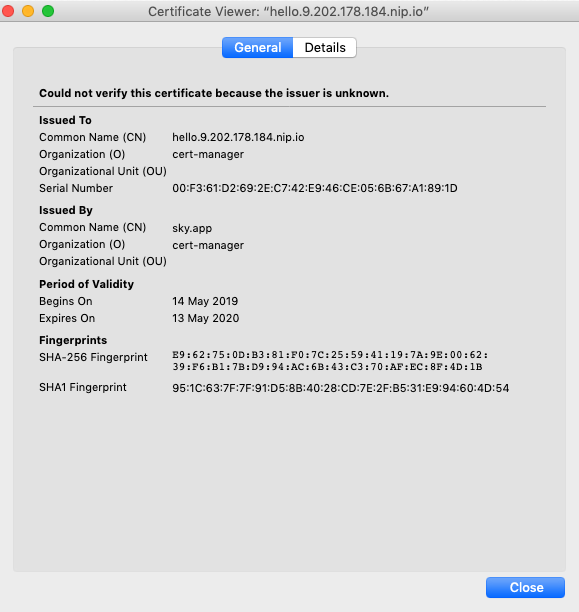
- backend:

serviceName: helloworld

servicePort: 5000

1. Access the application using <https://helloworld.x.x.x.x.nip.io/hello>

If you are using self-signed key, probably a warning message on the certificate would appear, try to view the detail of the certificate

****

**Summary**

You have gone through the steps of deploying Ingress with self-signed certificate generated through custom CA Issuer.

**Try the following**

* Enabling SSL with third-party certificates for IBM Cloud Private and deploy the Liberty server

<https://www.ibm.com/support/knowledgecenter/SSD28V_9.0.0/com.ibm.websphere.wlp.core.doc/ae/twlp_icp_auto_ssl3.html>

* Try to deploy liberty application with default ICP issuer

<https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/manage_applications/create_issuer.html>