Trustee Passport Mode

This document contains all the necessary details on setting up the trustee in passport mode. In the previous trustee setups i.e, BuiltIn and gRPC modes the deployments are controlled by the trustee-operator-controller. But it is not possible to set up the passport mode trustee using this operator because this operator is set up in such a way that we would be able to set up only one trustee deployment. If we try to deploy another trustee in the same namespace, this operator will overwrite the existing deployment instead of creating a new one. In our trustee setup, we have removed the dependency with this operator-controller and made sure that we set up the trustees by directly deploying the deployment and config files.

In passport mode, one trustee will have the AS and RVPS running inside it and the other won't. In order for another trustee to verify the token and release resources, the key used to sign the CoCo attestation token has to be mutually agreed. This allows one KBS to release resources given an attestation token signed by another trustee (KBS-AS). Before setting up the passport mode trustee, we need to generate the necessary token signature keys and certificates required for proper functioning of the overall trustee.

Generating Token Signing Keys and Certificates

1. First generate the CA Keys and Certificates required to generate the token signing keys and certificates later.

```
openssl genrsa -traditional -out ca.key 2048

openssl req -new -key ca.key -out ca-req.csr -subj
"/O=CNCF/OU=CoCo/CN=KBS-test-root"

openssl req -x509 -days 3650 -key ca.key -in ca-req.csr
-out ca-cert.pem
```

2. Now generate EC Private key. This creates an elliptic curve (EC) private key using the prime256v1 curve, saved as ***token.key***.

```
openssl ecparam -name prime256v1 -genkey -noout -out
token.key
```

3. Now create Certificate Signing Request (CSR) using the private key, with the specified subject fields.

```
openssl req -new -key token.key -out token-req.csr -subj
"/O=CNCF/OU=CoCo/CN=CoCo-AS"
```

4. Sign the CSR with CA (***ca-cert.pem*** and ***ca.key***), producing the token certificate (***token-cert.pem***).

```
openssl x509 -req -in token-req.csr -CA ca-cert.pem -CAkey
ca.key -CAcreateserial -out token-cert.pem
```

5. Create the certificate chain by concatenating the token certificate and CA certificate into a certificate chain file (***token-cert-chain.pem***).

```
cat token-cert.pem ca-cert.pem > token-cert-chain.pem
```

6. Create secrets for the following files required for parsing them in the config files of both trustees.

    a. token.key
    b. token-cert-chain.pem
    c. ca-cert.pem

```
kubectl create secret generic coco-token-key \
--from-file=token.key=./token.key \  -n operators

kubectl create secret generic coco-token-cert-chain \
--from-file=token-cert-chain.pem=./token-cert-chain.pem \  -n
operators

kubectl create secret generic coco-ca-cert \
--from-file=ca-cert.pem=./ca-cert.pem \  -n operators
```

Generate Certificates and Keys for both Trustees

1. Trustee 1 (KBS-AS)

Create a separate directory for trustee1 KBS-AS and trustee2 KBS in order to avoid confusion between the files used for both trustees during the setup.

```
mkdir trustee1
mkdir trustee2
cd trustee1
```

Authorization key-pair generation

First of all, we'd need to create the key pairs for Trustee authorization. The public key is used by the Trustee for client authorization, the private key is used by the client to prove its identity and register keys/secrets.

Create secret for client authorization:

```
openssl genpkey -algorithm ed25519 > privateKey

openssl pkey -in privateKey -pubout -out publicKey

kubectl create secret generic kbs-auth-public-key --from-file=publicKey
-n operators
```

NOTE: The keys used can be the same/different for both trustees here which doesn't matter for the functioning of the trustee setup.

HTTPS configuration

It is recommended to enable the HTTPS protocol for the following reasons:

- secure the Trustee server API
- bind the Trusted Execution Environment (TEE) to a given Trustee server by seeding the public key and certificate (as measured init data)

Instructions

1. In this example we're going to create a self-signed certificate using the following template:

```
cat << EOF > kbs-service-509.conf
[req]
default_bits       = 2048
default_keyfile    = localhost.key
distinguished_name = req_distinguished_name
req_extensions     = req_ext
x509_extensions    = v3_ca
[req_distinguished_name]
countryName                 = Country Name (2 letter code)
countryName_default         = UK
stateOrProvinceName         = State or Province Name (full name)
stateOrProvinceName_default = England
localityName                = Locality Name (eg, city)
localityName_default        = Bristol
```

```
organizationName            = Organization Name (eg, company)
organizationName_default    = Red Hat
organizationalUnitName      = organizationalunit
organizationalUnitName_default = Development
commonName                  = trustee1.iudx.io
commonName_default          = trustee1.iudx.io
commonName_max              = 64
[req_ext]
subjectAltName = @alt_names
[v3_ca]
subjectAltName = @alt_names
[alt_names]
DNS.1   = trustee1.iudx.io
EOF
```

2. Create secret for self-signed certificate:

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout
https.key -out https.crt \
    -config kbs-service-509.conf -passin pass:\
    -subj "/C=UK/ST=England/L=Bristol/O=Red
Hat/OU=Development/CN=trustee1.iudx.io"

kubectl create secret generic trustee1-https-certificate
--from-file=https.crt -n operators

kubectl create secret generic trustee1-https-key
--from-file=https.key -n operators
```

2. Trustee 2 (KBS)

Move to the trustee2 location for replicating the above files generated for trustee1 in trustee2.

```
cd ../trustee2
```

We can generate separate authorization keys for trustee2 but for simplicity, we are using the same keys for both trustees.

HTTPS configuration

The default DNS Public IP used for trustee2 will be changed from the above instructions. Everything else will remain the same.

Instructions

1. In this example we're going to create a self-signed certificate using the following template:

```
cat << EOF > kbs-service-509.conf
[req]
default_bits       = 2048
default_keyfile    = localhost.key
distinguished_name = req_distinguished_name
req_extensions     = req_ext
x509_extensions    = v3_ca
[req_distinguished_name]
countryName                 = Country Name (2 letter code)
countryName_default         = UK
stateOrProvinceName         = State or Province Name (full name)
stateOrProvinceName_default = England
localityName                = Locality Name (eg, city)
localityName_default        = Bristol
organizationName            = Organization Name (eg, company)
organizationName_default    = Red Hat
organizationalUnitName      = organizationalunit
organizationalUnitName_default = Development
commonName                  = trustee2.iudx.io
commonName_default          = trustee2.iudx.io
commonName_max              = 64
[req_ext]
subjectAltName = @alt_names
[v3_ca]
subjectAltName = @alt_names
[alt_names]
DNS.1   = trustee2.iudx.io
EOF
```

2. Create secret for self-signed certificate:

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout
https.key -out https.crt \
    -config kbs-service-509.conf -passin pass:\
    -subj "/C=UK/ST=England/L=Bristol/O=Red
Hat/OU=Development/CN=trustee2.iudx.io"

kubectl create secret generic trustee2-https-certificate
--from-file=https.crt -n operators

kubectl create secret generic trustee2-https-key
--from-file=https.key -n operators
```

Generate the necessary Config files

This command will create the ConfigMap object that provides the Trustee all the needed configuration.

KBS-Config

Unlike gRPC, we are going to use the builtin mode for setting up the config for trustee1. We can follow similar instructions if we want to set up for the gRPC mode by referring to this document tab but make sure that you set up the token signing key as provided in the following config files for gRPC KBS trustee as well.

```
cd ../trustee1

vim kbs-config.yaml
```

Place the following config information in the kbs-config.yaml file.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kbs-config-t1
  namespace: operators
data:
  kbs-config.toml: |
    [http_server]
    sockets = ["0.0.0.0:8080"]
    insecure_http = false
    private_key = "/etc/https-key/https.key"
    certificate = "/etc/https-cert/https.crt"
    [admin]
    insecure_api = true
    auth_public_key = "/etc/auth-secret/publicKey"
    [attestation_token]
    insecure_key = true
    attestation_token_type = "CoCo"
    [attestation_service]
    type = "coco_as_builtin"
    work_dir = "/opt/confidential-containers/attestation-service"
    policy_engine = "opa"
      [attestation_service.attestation_token_broker]
      type = "Ear"
      policy_dir =
"/opt/confidential-containers/attestation-service/policies"
        [attestation_service.attestation_token_broker.signer]
          key_path = "/etc/coco-token-key/token.key"
```

```
          cert_path =
"/etc/coco-token-cert-chain/token-cert-chain.pem"
      [attestation_service.attestation_token_config]
      duration_min = 5
      [attestation_service.rvps_config]
      type = "BuiltIn"

        [attestation_service.rvps_config.storage]
        type = "LocalJson"
        file_path =
"/opt/confidential-containers/rvps/reference-values/reference-val
ues.json"
    [[plugins]]
    name = "resource"
    type = "LocalFs"
    dir_path = "/opt/confidential-containers/kbs/repository"
    [policy_engine]
    policy_path = "/opt/confidential-containers/opa/policy.rego"
```

```
kubectl apply -f kbs-config.yaml -n operators
```

Here, you can observe that we did place the token signing key and certificates used for signing the token after verifying the hardware evidence. Make sure that this configuration is set up if you want to deploy for gRPC mode as well.

Now for trustee2, the following configuration file would be used. As the 2nd trustee has nothing to do with the AS and RVPS, we have removed the information from the config file. Here, you can observe that we have placed CA-Certificate in the following file which helps us to verify the token. If this CA matches with the cert chain and signing key above, you should be all set.

```
cd ../trustee2

vim kbs-config.yaml
```

Place the following config information in the kbs-config.yaml file.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kbs-config-t2
  namespace: operators
data:
  kbs-config.toml: |
```

```
    [http_server]
    sockets = ["0.0.0.0:8080"]
    insecure_http = false
    private_key = "/etc/https-key/https.key"
    certificate = "/etc/https-cert/https.crt"
    [admin]
    insecure_api = true
    auth_public_key = "/etc/auth-secret/publicKey"
    [attestation_token]
    trusted_certs_paths = ["/etc/coco-ca-cert/ca-cert.pem"]
    insecure_key = false
    [[plugins]]
    name = "resource"
    type = "LocalFs"
    dir_path = "/opt/confidential-containers/kbs/repository"
    [policy_engine]
    policy_path = "/opt/confidential-containers/opa/policy.rego"
```

```
kubectl apply -f kbs-config.yaml -n operators
```

Reference Values

The reference values are an important part of the attestation process. The client collects the measurements (from the running software, the TEE hardware and its firmware) and submits a quote with the claims to the attestation server. These measurements, in order for the attestation protocol to succeed, have to match one of potentially multiple configured valid values that had been registered to Trustee previously. You could also apply flexible rules like “firmware of secure processor > v1.30”, etc. This process guarantees the cVM (confidential VM) is running the expected software stack and that it hasn’t been tampered with.

NOTE: According to your HW platform, you’d need to register some real trusted digests of the podVM in the reference values. In our case, we have used 3, 8, 9 and 11 reference values. CoCo CNCF members have informed us via Slack channel that 3, 9 and 11 are good fit for the podVM and PCR 8 value is for the initdata.

```
cd ../trustee1

vim rvps-reference-values-passport.yaml
```

Place the following information in the file.

```
apiVersion: v1
```

```
kind: ConfigMap
metadata:
  name: rvps-reference-values-t1
  namespace: operators
data:
  reference-values.json: |
    [
      {
        "name": "snp",
        "expiration": "2026-01-01T00:00:00Z",
        "hash-value": [
          {
            "alg": "sha256",
            "value":
"0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E79
69"
          },
          {
            "alg": "sha256",
            "value":
"0x1FDA9EBE9BE062EEAB483C35505B48A4558FB1C614B8F42733DD8470EA78DB
A9"
          },
          {
            "alg": "sha256",
            "value":
"0x98F559EAB61223E7C85EC7C8FF28B2002D7DAA794D3D379975616B7E9790F3
76"
          },
          {
            "alg": "sha256",
            "value":
"0x639CCB858044702A42E7CE73169B08293F63FABFB9999BC40927FD860413CF
98"
          }
        ]
      }
    ]
```

```
kubectl apply -f rvps-reference-values-passport.yaml -n operators
```

Create Resource Policy

The resource policy can be injected in the trustee configuration by creating a ConfigMap. Here, the chosen ConfigMap name has to be part of the KbsConfig CR definition.

In the following example, the policy is very permissive and allows each client to retrieve a secret/key without checking for the EAR attestation token. In a production environment it is recommended to check the content of the EAR token and authorize the client only if the *ear.status* is not *contraindicated*.

```
vim resource-policies-passport.yaml
```

Place the following information in the file.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: resource-policy-t1
  namespace: operators
data:
  policy.rego:
    package policy
    default allow = true
```

```
kubectl apply -f resource-policies-passport.yaml -n operators
```

Generate the Services and Ingress Controller for both Trustees

1. Generate the following services for both the trustees.

   ```
   cd ../trustee1

   vim kbs-as-svc.yaml
   ```

   Place the following code inside this file.

   ```
   apiVersion: v1
   kind: Service
   metadata:
     name: trustee-as-kbs-service
   ```

```
  namespace: operators
spec:
  selector:
    app: trustee-as-kbs
  ports:
    - protocol: TCP
      port: 8080       # Exposed on public IP
      targetPort: 8080 # Inside the pod/container
  type: LoadBalancer
```

```
kubectl apply -f kbs-as-svc.yaml -n operators
```

2. Do the same for trustee2 as well.

```
cd ../trustee2

vim kbs-svc.yaml
```

Place the following code inside this file.

```
apiVersion: v1
kind: Service
metadata:
  name: trustee-kbs-service
  namespace: operators
spec:
  selector:
    app: trustee-kbs
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 8080          # Exposed on public IP
      targetPort: 8080    # Inside the pod/container
```

```
kubectl apply -f kbs-svc.yaml -n operators
```

3. Create the ingress controller for both these trustees which helps us to generate the external IP for both these trustees which are static in nature.

```
cd ../

vim ingress.yaml
```

Place the following code inside this file.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: kbs-ingress
  namespace: operators
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
  - host: trustee1.iudx.io
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: trustee-as-kbs-service
            port:
              number: 80
  - host: trustee2.iudx.io
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: trustee-kbs-service
            port:
              number: 80
```

```
kubectl apply -f ingress.yaml -n operators
```

Deploying both Trustees

1. Create the following deployment file for Trustee1 (KBS + AS) required to set up the pod for this trustee.

```
cd ../trustee1

vim kbs-as.yaml
```

Place the following code inside the yaml file.

```
apiVersion: v1
kind: Pod
metadata:
  name: trustee-as-kbs
  labels:
    app: trustee-as-kbs
  namespace: operators
spec:
  containers:
  - command:
    - /usr/local/bin/kbs
    - --config-file
    - /etc/kbs-config/kbs-config.toml
    image:
ghcr.io/confidential-containers/key-broker-service:built-in-as-v0
.11.0
    imagePullPolicy: IfNotPresent
    name: kbs
    ports:
    - containerPort: 8080
      name: kbs
      protocol: TCP
    volumeMounts:
    - mountPath: /opt/confidential-containers
      name: confidential-containers
    - mountPath:
/opt/confidential-containers/kbs/repository/default
      name: default
    - mountPath: /etc/kbs-config
      name: kbs-config
    - mountPath: /opt/confidential-containers/opa
      name: opa
    - mountPath: /etc/auth-secret
      name: auth-secret
    - mountPath: /etc/https-key
      name: https-key
    - mountPath: /etc/https-cert
      name: https-cert
    - mountPath: /etc/coco-token-cert-chain
      name: coco-token-cert-chain
    - mountPath: /etc/coco-token-key
      name: coco-token-key
    - mountPath:
/opt/confidential-containers/rvps/reference-values
      name: reference-values
  restartPolicy: Always
```

```
  volumes:
  - emptyDir:
      medium: Memory
    name: confidential-containers
  - emptyDir:
      medium: Memory
    name: default
  - configMap:
      defaultMode: 420
      name: kbs-config-t1
    name: kbs-config
  - configMap:
      defaultMode: 420
      name: resource-policy-t1
    name: opa
  - name: auth-secret
    secret:
      defaultMode: 420
      secretName: kbs-auth-public-key
  - name: https-key
    secret:
      defaultMode: 420
      secretName: trustee1-https-key
  - name: https-cert
    secret:
      defaultMode: 420
      secretName: trustee1-https-certificate
  - name: coco-token-cert-chain
    secret:
      defaultMode: 420
      secretName: coco-token-cert-chain
  - name: coco-token-key
    secret:
      defaultMode: 420
      secretName: coco-token-key
  - configMap:
      defaultMode: 420
      name: rvps-reference-values-t1
    name: reference-values
```

```
kubectl apply -f kbs-as.yaml -n operators
```

2. Similarly do the same for the trustee 2 (KBS without AS).

```
cd ../trustee2

vim kbs.yaml
```

Place the following code inside the yaml file.

```
apiVersion: v1
kind: Pod
metadata:
  name: trustee-kbs
  labels:
    app: trustee-kbs
  namespace: operators
spec:
  containers:
  - command:
    - /usr/local/bin/kbs
    - --config-file
    - /etc/kbs-config/kbs-config.toml
    image:
ghcr.io/confidential-containers/key-broker-service:built-in-as-v0
.11.0
    imagePullPolicy: IfNotPresent
    name: kbs
    ports:
    - containerPort: 8080
      name: kbs
      protocol: TCP
    volumeMounts:
    - mountPath: /opt/confidential-containers
      name: confidential-containers
    - mountPath:
/opt/confidential-containers/kbs/repository/default
      name: default
    - mountPath: /etc/kbs-config
      name: kbs-config
    - mountPath: /opt/confidential-containers/opa
      name: opa
    - mountPath: /etc/auth-secret
      name: auth-secret
    - mountPath: /etc/https-key
      name: https-key
    - mountPath: /etc/https-cert
      name: https-cert
    - mountPath: /etc/coco-ca-cert
      name: coco-ca-cert
```

```
  restartPolicy: Always
  volumes:
  - emptyDir:
      medium: Memory
    name: confidential-containers
  - emptyDir:
      medium: Memory
    name: default
  - configMap:
      defaultMode: 420
      name: kbs-config
    name: kbs-config
  - configMap:
      defaultMode: 420
      name: resource-policy
    name: opa
  - name: auth-secret
    secret:
      defaultMode: 420
      secretName: kbs-auth-public-key
  - name: https-key
    secret:
      defaultMode: 420
      secretName: trustee2-https-key
  - name: https-cert
    secret:
      defaultMode: 420
      secretName: trustee2-https-certificate
  - name: coco-ca-cert
    secret:
      defaultMode: 420
      secretName: coco-ca-cert
```

```
kubectl apply -f kbs.yaml -n operators
```

3. Now check if both the pods are running or not.

kubectl get pods -n operators

```
# You should get the output as follows

NAME                READY   STATUS    RESTARTS   AGE
trustee-as-kbs      1/1     Running   0          7d4h
trustee-kbs         1/1     Running   0          7d3h
```

4. We can check the logs in both the trustees as well to make sure the passport mechanism is running well and fine.

Initdata Configuration

Here, initdata file is used to inform the workload pod the necessary URL required to connect with the trustees. There are 2 sections in this initdata. One is for Attestation Agent (AA) and other is for Confidential Data Hub (CDH). In AA, we place all the information required for connecting AA with the Trustee1 (KBS + AS) and CDH stores all the information required for connecting CDH with the Trustee2 (KBS without AS).

In AA, urls can be replaced with the public IP of Trustee1 i.e. trustee1.iudx.io and cert is replaced with the https.crt placed in the trustee1 directory. Similarly for CDH, we replace url with the public IP of Trustee2 i.e. trustee2.iudx.io and cert is replaced with the https.crt placed in the trustee2 directory.

```
algorithm = "sha256"
version = "0.1.1"
[data]
"aa.toml" = '''
[token_configs]
[token_configs.coco_as]
url = 'https://trustee1.iudx.io:8080'
[token_configs.kbs]
url = 'https://trustee1.iudx.io:8080'
cert = """
-----BEGIN CERTIFICATE-----
MIIDtDCCApygAwIBAgIUVCgg0qruAgipPQmiUIzcaMxnjV4wDQYJKoZIhvcNAQEL
BQAwdDELMAkGA1UEBhMCVUsxEDAOBgNVBAgMB0VuZ2xhbmQxEDAOBgNVBAcMB0Jy
aXN0b2wxEDAOBgNVBAoMB1JlZCBIYXQxFDASBgNVBAsMC0RldmVsb3BtZW50MRkw
FwYDVQQDDBB0cnVzdGVlMS5pdWR4LmlvMB4XDTI1MDUxNDA2MTE0N1oXDTI2MDUx
NDA2MTE0N1owdDELMAkGA1UEBhMCVUsxEDAOBgNVBAgMB0VuZ2xhbmQxEDAOBgNV
BAcMB0JyaXN0b2wxEDAOBgNVBAoMB1JlZCBIYXQxFDASBgNVBAsMC0RldmVsb3Bt
ZW50MRkwFwYDVQQDDBB0cnVzdGVlMS5pdWR4LmlvMIIBIjANBgkqhkiG9w0BAQEF
AAOCAQ8AMIIBCgKCAQEAtDqb7AcsCnUMSy6JQYyiSgiZwlJsTFQdx0rvGlyIhdt6
d1WwHHsytIoHFR3v/ME62FLY775V7QT1UM4X4iKe7wTkK4nahSvrYvKKi43IVHrq
ksFiHFU/UrwYI2ZnvaBCWfquJHTBPnuhjsh0tV689E+5Q8dvhOGBBJ2sQE4UQc9u
hx4FKid3qIo9kRlxVD3ni3029YVfXr+I+sW6jWOh1548ipjegmkaAtu7IQi+f+XR
lUn6BIH3YSVfEihKLSY41EDwlMNNFvaETbgF7DN5ipJ+p6H+M66JuwID6No086QO
/ALZrWQHPKka1aSzAZ2iSyWyYdVPT283x3+pRwKY8wIDAQABoz4wPDAbBgNVHREE
FDASghB0cnVzdGVlMS5pdWR4LmlvMB0GA1UdDgQWBBSLOYnK7DYOuzShDNTs3bLs
Jt/WSzANBgkqhkiG9w0BAQsFAAOCAQEABTkmE/4L7blDkhu79F0xbBUzGzA4S1mz
VF0uNG1XDBjzElL5prQQuKRxSEN5nDyO93UTEGMQ2uPlUy6SlmOfxvKYc/djrbSt
4DtPtadbJkV6QIRm6x9C/47rGlTdV+Fo7OkJaG6iMG232WwKDlu4MQb29S64fbkR
dWXccVl1DftA/hEgIbs2a4CmbH+X3U+rsvKkbx3jnL7rihicDN/UkGLP4zgb/iTB
h551iYk/M0PQ6YlzvdaF7tRAVqJXYt+edIWBkmXJZcywhmWI+LzVklle5Dku7trN
```

```
wzqZN7n074mXtsX74Kdv7twoF9NnvE5OGUQwkbjsFl62qCjlImVbEA==
-----END CERTIFICATE-----
"""
'''
"cdh.toml"  = '''
socket = 'unix:///run/confidential-containers/cdh.sock'
[kbc]
name = 'cc_kbc'
url = 'https://trustee2.iudx.io:8080'
kbs_cert = """
-----BEGIN CERTIFICATE-----
MIIDtDCCApygAwIBAgIUTvsP95g9vBqfis9LASPN6C5YXwUwDQYJKoZIhvcNAQEL
BQAwdDELMAkGA1UEBhMCVUsxEDAOBgNVBAgMB0VuZ2xhbmQxEDAOBgNVBAcMB0Jy
aXN0b2wxEDAOBgNVBAoMB1JlZCBIYXQxFDASBgNVBAsMC0RldmVsb3BtZW50MRkw
FwYDVQQDDBB0cnVzdGVlMi5pdWR4LmlvMB4XDTI1MDUxNDA2MTQyMloXDTI2MDUx
NDA2MTQyMlowdDELMAkGA1UEBhMCVUsxEDAOBgNVBAgMB0VuZ2xhbmQxEDAOBgNV
BAcMB0JyaXN0b2wxEDAOBgNVBAoMB1JlZCBIYXQxFDASBgNVBAsMC0RldmVsb3Bt
ZW50MRkwFwYDVQQDDBB0cnVzdGVlMi5pdWR4LmlvMIIBIjANBgkqhkiG9w0BAQEF
AAOCAQ8AMIIBCgKCAQEAqeuW6fzPIIy15hHvQ9QG7nC6hrZiMq/a9wIHgtNTxWRs
PSV5aMpE4eHxnkxSqX2HQngKfapl/xW9dPFzfav9T9ZedKxub7mWkMARWf2ISOr8
vz9rSkjt61Vseo7gXHVAlFvjWvbDlMcfRnkG5cMZDqm+XJqCPA54vYLYMYBlfFvX
weAGC+NAMmYUVAAp647SZkip6iLMjzlKG9EGoCkbnO7crN5+79OZ2LgHfJicd63+
kBLzW4vYTmb7A4/2tYLDGzIHK4wfXa6TNVjaykGViPBkNTXGcAKl9303AhL7KdhT
THe8/X5vQvonsjySTTeMZVYHq4drauNyGI38OROmGQIDAQABoz4wPDAbBgNVHREE
FDASghB0cnVzdGVlMi5pdWR4LmlvMB0GA1UdDgQWBBQ8bJb+Vb1eCFANK5lOfdlj
SRDr/DANBgkqhkiG9w0BAQsFAAOCAQEAnjJW9cgAPCxDfUkaCznxRD48NgYZRt4c
pwcN2P7C4oiNgK4XF65m2DnB2BvKsCgsEr+y7vPftoQeTP3OUHnsUPxJxlswqciT
7Y89fHTZBt2AZGhFwCEBPb2x0E7JSS6ffDy4fObHElAGOwl1dPFz6ff4mdALnDFC
0+xt3uf3AfLVYwpO8MSzGhHmbW72kuc1NDdlNA0fhuhTUbQK1FxmuYyAU629B8wX
EW056qpARF6Y6wimAO7Jx5PmFl9qNuIOwLgXlpJZuQzI8W+6mShhk/xNNDP0mEZK
OlMpYLsb3b6Y2I36JaSXFaOSO8vd8pUgNJ0rO4jfajsDPeUQqES1Rw==
-----END CERTIFICATE-----
"""
'''
```

Rest all steps required to run the workload pod will remain the same as provided in the [document tab](#).

Some Important Commands to Note

1. When we want to run the trustee pods if they don't appear/ not in running state, delete the pods and run the following command.

```
cd trustee1

kubectl apply -f kbs-as.yaml -n operators

cd ../trustee2

kubectl apply -f kbs.yaml -n operators
```

2. When we want to run the nginx workload pod, run these commands assuming the nginx-encrypted image is already present in the x86 machine.

```
export KEY_FILE="coco_image_key"

export KEY_PATH="/default/coco_image_key/test_image"

kubectl exec -n operators trustee-kbs -- mkdir -p
"/opt/confidential-containers/kbs/repository/$(dirname
"$KEY_PATH")"

cat "$KEY_FILE" | kubectl exec -i -n operators trustee-kbs
-c kbs -- tee
"/opt/confidential-containers/kbs/repository/${KEY_PATH}" >
/dev/null

cd ~/coco-v12/trustee-passport/nginx-encrypted

kubectl apply -f nginx-encrypted.yaml -n operators
```