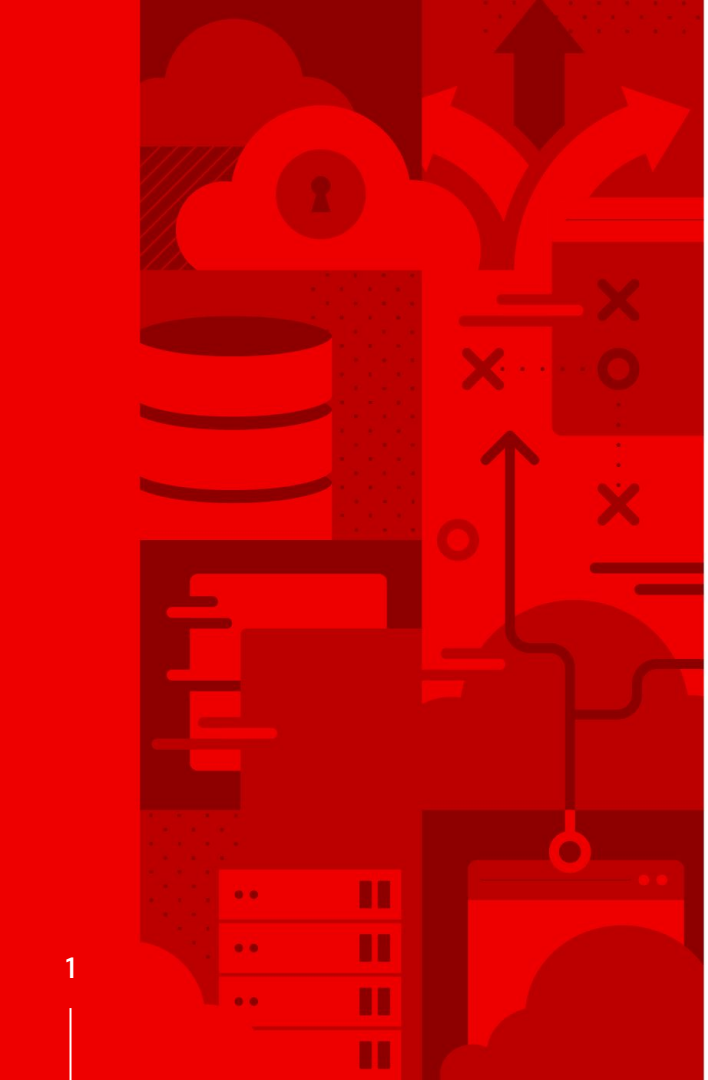


OpenShift 4 TLS Certificates

Presentation by:

Mohit and Ayush

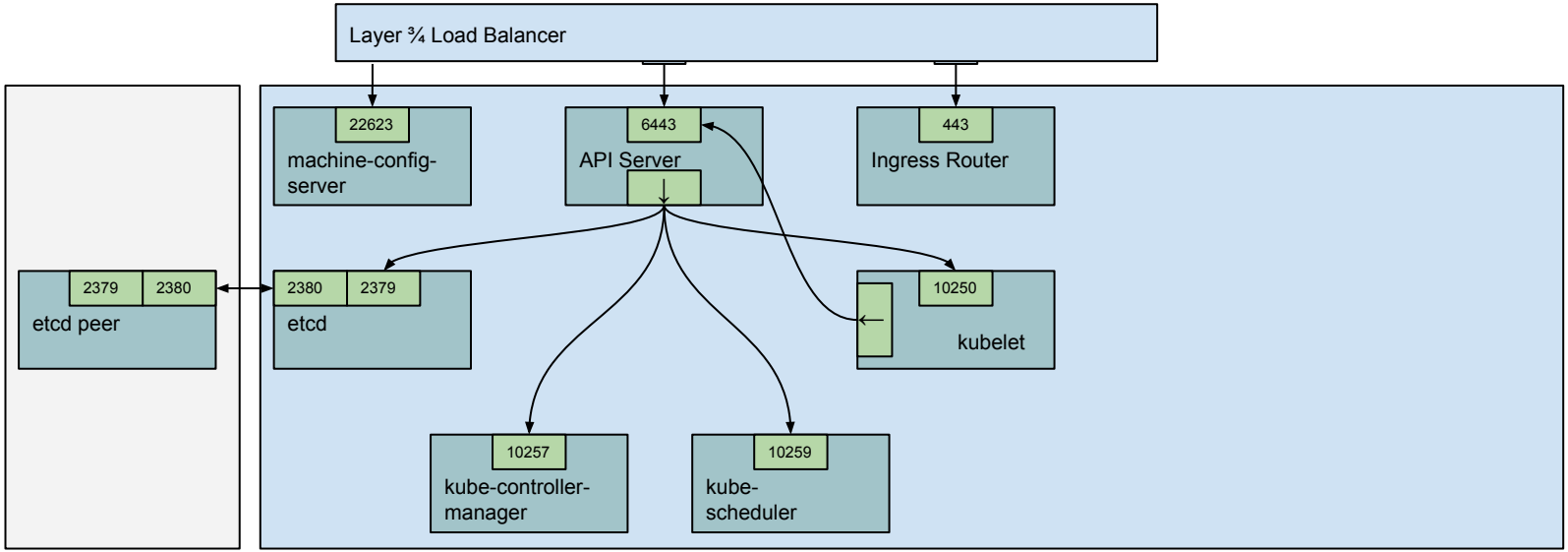
Red Hat

Agenda

What we'll discuss today

- Important Certificate
- Control Plane Certificate
- Kubelet Certificate
- TLS Node Bootstrapping
- Certificate Authorities
- Certificates Recovery
- Cert Rotation
- Ingress Controller Certificate
- Cluster-wide Proxy
- Known Proxy Issue
- Adding Custom CA to pods
- Troubleshooting

Important certs in OpenShift



Red Hat

Control Plane Certificates

Control plane certificates are managed by the system and rotated automatically.

`Api` Certificate - the certificate presented by the API server for external requests

`Api-int` Certificate - the certificate presented by the API server for cluster-internal communication

Retrieve Certificates from the secrets and the location on masters.

```
# oc get secret -n openshift-kube-apiserver external-loadbalancer-serving-certkey -o yaml -o=custom-columns=":.data.tls\.crt" | tail -1 | base64 -d | openssl x509 -text
# openssl x509 -in  /etc/kubernetes/static-pod-resources/kube-apiserver-certs/secrets/external-loadbalancer-serving-certkey/tls.crt -text

# oc get secret -n openshift-kube-apiserver internal-loadbalancer-serving-certkey -o yaml -o=custom-columns=":.data.tls\.crt" | tail -1 | base64 -d | openssl x509 -text
# openssl x509 -in  /etc/kubernetes/static-pod-resources/kube-apiserver-certs/secrets/internal-loadbalancer-serving-certkey/tls.crt -text
```

https://github.com/openshift/cluster-kube-apiserver-operator/blob/0b686ff00295c382f245b0b4103a566d672498c8/pkg/operator/certrotationcontroller/certrotationcontroller.go

Add an API server named certificate

https://docs.openshift.com/container-platform/4.4/authentication/certificates/api-server.html

Control Plane Certificates

Kube Controller Manager and Kube Scheduler

The serving cert is valid for 2 years and client certs are valid for 30 days & get rotated automatically

Retrieve Certificates from the secrets and the location on masters.

```
# openssl x509 -in /etc/kubernetes/static-pod-resources/kube-controller-manager-certs/secrets/kube-controller-manager-client-cert-key/tls.crt -text
# oc get secret kube-controller-manager-client-cert-key -n openshift-kube-controller-manager -o=custom-columns=":.data.tls\.crt" | tail -1 | base64 -d | openssl x509 -text

# openssl x509 -in /etc/kubernetes/static-pod-resources/kube-controller-manager-pod-7/secrets/serving-cert/tls.crt -text
# oc get secret serving-cert -n openshift-kube-controller-manager -o=custom-columns=":.data.tls\.crt" | tail -1 | base64 -d | openssl x509 -text

# openssl x509 -in /etc/kubernetes/static-pod-resources/kube-scheduler-certs/secrets/kube-scheduler-client-cert-key/tls.crt -text
# oc get secret kube-scheduler-client-cert-key -n openshift-kube-scheduler -o=custom-columns=":.data.tls\.crt" | tail -1 | base64 -d | openssl x509 -text

# openssl x509 -in /etc/kubernetes/static-pod-resources/kube-scheduler-pod-8/secrets/serving-cert/tls.crt -text
# oc get secret serving-cert -n openshift-kube-scheduler -o=custom-columns=":.data.tls\.crt" | tail -1 | base64 -d | openssl x509 -text
```

https://github.com/openshift/cluster-kube-apiserver-operator/blob/0b686ff00295c382f245b0b4103a566d672498c8/pkg/operator/certrotationcontroller/certrotationcontroller.go

Control Plane Certificates

ETCD Certificates

etcd certificates are signed by the etcd-signer; they come from a certificate authority (CA) that is generated by the bootstrap process.
Metric certificates: All metric consumers connect to proxy with metric-client certificates.
etcd-client, etcd-metric-client, etcd-metric-signer, and etcd-signer) are added to the openshift-config, openshift-monitoring, and openshift-kube-apiserver namespaces.

All etcd certificates are valid for 10 years.

ETCD Root CA
openssl x509 -in /etc/kubernetes/static-pod-resources/etcd-member/root-ca.crt -text

ETCD CA which signs serving and peer certificate
openssl x509 -in /etc/kubernetes/static-pod-resources/etcd-member/ca.crt -text

ETCD Metrics CA which sign metrics certificate
openssl x509 -in /etc/kubernetes/static-pod-resources/etcd-member/metric-ca.crt -text

Control Plane Certificates

ETCD Certificate

ETCD Serving Cert
openssl x509 -in /etc/kubernetes/static-pod-resources/etcd-certs/secrets/etcd-all-serving/etcd-serving-master-0.mohittestocp43.lab.pnq2.cee.redhat.com.crt -text

ETCD Peer Cert
openssl x509 -in /etc/kubernetes/static-pod-resources/etcd-certs/secrets/etcd-all-peer/etcd-peer-master-0.mohittestocp43.lab.pnq2.cee.redhat.com.crt -text

ETCD Serving Metrics Cert
openssl x509 -in /etc/kubernetes/static-pod-resources/etcd-certs/secrets/etcd-all-serving-metrics/etcd-serving-metrics-master-0.mohittestocp43.lab.pnq2.cee.redhat.com.crt -text

Kubelet Certificate

Client certificates for the kubelet to authenticate to the API server.
Kubelet serving certificate.

The node certificates are valid for 30 days and gets auto refreshed.

```
# openssl x509 -in /var/lib/kubelet/pki/kubelet-client-current.pem -text
```

```
# openssl x509 -in /var/lib/kubelet/pki/kubelet-server-current.pem -text
```

TLS Node Bootstrapping

1. kubelet begins
2. kubelet sees that it does not have a kubeconfig file
3. kubelet searches for and finds a bootstrap-kubeconfig file
4. kubelet reads its bootstrap file, retrieving the URL of the API server and a limited usage "token"
5. kubelet connects to the API server, authenticates using the token
6. kubelet now has limited credentials to create and retrieve a certificate signing request (CSR)
7. kubelet creates a CSR for itself with the signerName set to kubernetes.io/kube-apiserver-client-kubelet
8. CSR is approved in one of two ways:
9. If configured, kube-controller-manager automatically approves the CSR
10. If configured, an outside process, possibly a person, approves the CSR using the Kubernetes API or via kubectl
11. Certificate is created for the kubelet
12. Certificate is issued to the kubelet
13. kubelet retrieves the certificate
14. kubelet creates a proper kubeconfig with the key and signed certificate
15. kubelet begins normal operation
16. Optional: if configured, kubelet automatically requests renewal of the certificate when it is close to expiry
17. The renewed certificate is approved and issued, either automatically or manually, depending on configuration.

Node certificates are expired or having some other issue

1. We need to create a bootstrap kubeconfig . To do so, run the following from a place with cluster-admin access:

oc -n openshift-machine-config-operator serviceaccounts create-kubeconfig node-bootstrapper > kubeconfig

2. SSH to the master and become root

3. Stop kubelet: systemctl stop kubelet

4. Backup and remove the following files:

mv /var/lib/kubelet/kubeconfig{,.old}
mv /var/lib/kubelet/pki{,.old}
mkdir /var/lib/kubelet/pki

5. Copy the bootstrap kubeconfig created at step 1 to /etc/kubernetes/kubeconfig

6. Start kubelet: systemctl start kubelet

7. In a place with cluster-admin access to the cluster, check whether new CSRs are generated (via "oc get csr --all-namespaces") and approve those which are not approved automatically (via "oc adm certificate approve <csr-name>"). At least 2 CSRs must be generated (each one may take some minutes)

Certificate Authority in OCP 4

- OpenShift creates it's own CA and signs it's own certs for ingress and API services. In OpenShift 3.x you are able to provide a custom certificate authority, but not in 4.x.
- In 4.x, only certificate authorities known to RHCOS will be used to validate the authenticity of a certificate.
- What are the known certificate authorities in RHCOS?

```
# openssl crl2pkcs7 -nocrl -certfile /etc/ssl/certs/ca-bundle.crt | openssl pkcs7 -print_certs -text
-noout
```

Certificate Authority in OCP 4

sa-token-signing-certs - this configmaps holds the certs used to verify the SA token JWTs created by the kube-controller-manager-operator

Kube-apiserver-aggregator-client-ca - this ca bundle contains certs to verify the aggregator. We copy it from the shared location to here.

csr-controller-ca - this configmap allows us to verify the kubelet serving certs

Kube-apiserver-client-ca - this ca bundle contains certs used by the kube-apiserver to verify client certs

Kubelet-serving-ca - this ca bundle contains certs that can be used to verify a kubelet

Kube-apiserver-server-ca - this ca bundle contains certs that can be used to verify a kube-apiserver

Bound-sa-token-signing-certs - this ca bundle contains public keys that can be used to verify bound tokens issued by the kube-apiserver

Admin-kubeconfig-client-ca - this is from the installer and contains the value to verify the admin.kubeconfig user

Kube-apiserver-to-kubelet-client-ca - this is from the installer and contains the value to verify the kube-apiserver communicating to the kubelet

Kube-control-plane-signer-ca - this bundle is what this operator uses to mint new client certs it directly manages

User-client-ca - this bundle is what a user uses to mint new client certs it directly manages

Kubelet-bootstrap-kubeconfig - this bundle is what validates the master kubelet bootstrap credential. Users can invalid this by removing it.

Loadbalancer-serving-ca - this bundle is what this operator uses to mint loadbalancers certs

Localhost-serving-ca - this bundle is what this operator uses to mint localhost certs

Service-network-serving-ca - this bundle is what a user uses to mint service-network certs

Localhost-recovery-serving-ca - this bundle is what this operator uses to mint localhost-recovery certs

https://github.com/openshift/cluster-kube-apiserver-operator/blob/master/pkg/operator/resourcesynccontroller/resourcesynccontroller.go

Control Plane Certificates Expired

https://github.com/openshift/cluster-kube-apiserver-operator/blob/master/pkg/recovery/apiserver.go
https://github.com/openshift/cluster-kube-apiserver-operator/blob/release-4.4/pkg/operator/certrotationcontroller/certrotationcontroller.go
https://github.com/openshift/cluster-kube-apiserver-operator/tree/master/bindata/v4.1.0/kube-apiserver

OpenShift clusters now automatically recover control plane certificates

We will take our existing `cluster-kube-apiserver-operator regenerated-certificates` command and create a simple controller which will watch for expired certificates and regenerate them. It will connect to the kube-apiserver using localhost with an SNI name option wired to a 10 year cert. When there is no work to do, this controller will do nothing. This controller will run as another container in our existing static pods

https://github.com/openshift/enhancements/blob/8ea26f5/enhancements/kube-apiserver/auto-cert-recovery.md

Cert Rotation

I0617 06:47:53.717332 1 event.go:281] Event(v1.ObjectReference{Kind:"Deployment", Namespace:"openshift-kube-apiserver-operator", Name:"kube-apiserver-operator", UID:"591508d0-95fd-40eb-94fe-7a464e730160", APIVersion:"apps/v1", ResourceVersion:"", FieldPath:""}): type: 'Normal' reason: 'TargetUpdateRequired' "kube-scheduler-client-cert-key" in "openshift-config-managed" requires a new target cert/key pair: past its refresh time 2020-06-17 06:47:48 +0000 UTC

I0617 06:47:53.719331 1 event.go:281] Event(v1.ObjectReference{Kind:"Deployment", Namespace:"openshift-kube-apiserver-operator", Name:"kube-apiserver-operator", UID:"591508d0-95fd-40eb-94fe-7a464e730160", APIVersion:"apps/v1", ResourceVersion:"", FieldPath:""}): type: 'Normal' reason: 'TargetUpdateRequired' "kube-controller-manager-client-cert-key" in "openshift-config-managed" requires a new target cert/key pair: past its refresh time 2020-06-17 06:47:48 +0000 UTC

I0617 06:47:53.864553 1 event.go:281] Event(v1.ObjectReference{Kind:"Deployment", Namespace:"openshift-kube-apiserver-operator", Name:"kube-apiserver-operator", UID:"591508d0-95fd-40eb-94fe-7a464e730160", APIVersion:"apps/v1", ResourceVersion:"", FieldPath:""}): type: 'Normal' reason: 'SecretUpdated' Updated Secret/kube-controller-manager-client-cert-key -n openshift-config-managed because it changed

I0617 06:47:54.922341 1 event.go:281] Event(v1.ObjectReference{Kind:"Deployment", Namespace:"openshift-kube-apiserver-operator", Name:"kube-apiserver-operator", UID:"591508d0-95fd-40eb-94fe-7a464e730160", APIVersion:"apps/v1", ResourceVersion:"", FieldPath:""}): type: 'Normal' reason: 'SecretUpdated' Updated Secret/kube-scheduler-client-cert-key -n openshift-config-managed because it changed

I0617 06:48:03.726965 1 event.go:281] Event(v1.ObjectReference{Kind:"Deployment", Namespace:"openshift-kube-apiserver-operator", Name:"kube-apiserver-operator", UID:"591508d0-95fd-40eb-94fe-7a464e730160", APIVersion:"apps/v1", ResourceVersion:"", FieldPath:""}): type: 'Normal' reason: 'TargetUpdateRequired' "external-loadbalancer-serving-certkey" in "openshift-kube-apiserver" requires a new target cert/key pair: past its refresh time 2020-06-17 06:48:02 +0000 UTC

https://github.com/openshift/cluster-kube-apiserver-operator/blob/master/vendor/k8s.io/client-go/tools/record/event.go

Verify API Certs

Verifying the public API Certificate:

```
# oc get secret -n openshift-kube-apiserver-operator loadbalancer-serving-signer -o=jsonpath='{.data.tls\.crt}' | base64 -d > loadbalancer-serving-signer.crt
# openssl s_client -showcerts -connect api.mohittestocp43.lab.pnq2.cee.redhat.com:6443 -CAfile loadbalancer-serving-signer.crt -servername api.mohittestocp43.lab.pnq2.cee.redhat.com:6443
```

Verifying the public API Certificate with a kubeconfig:

```
# cat  $KUBECONFIG | grep -P certificate-authority-data | cut -d ':' -f2 |sed 's/^ *//'  | base64 -d > kubeconfig-ca.crt
# openssl s_client -showcerts -connect api.mohittestocp43.lab.pnq2.cee.redhat.com:6443 -CAfile kubeconfig-ca.crt -servername api.mohittestocp43.lab.pnq2.cee.redhat.com
```

Control Plane Cert issue on 1 master node.

To force deployment of kubeaapiserver pods run below command :

oc patch kubeapiserver/cluster --type merge -p "{\"spec\":{\"forceRedeploymentReason\":\"Forcing new revision with random number $RANDOM to make message unique\"}}"

Similarly for kubecontroller manager :

oc patch kubecontrollermanager cluster -p='{"spec": {"forceRedeploymentReason": "recovery-"$(date --rfc-3339=ns)""}}' --type=merge

Kubescheduler :

oc patch kubescheduler cluster -p='{"spec": {"forceRedeploymentReason": "recovery-"$(date --rfc-3339=ns)""}}' --type=merge"

Installer Controller

https://github.com/openshift/cluster-kube-apiserver-operator/blob/master/vendor/github.com/openshift/library-go/pkg/operator/staticpod/controller/installer/installer_controller.go

Installer Pod yaml

https://github.com/openshift/origin/blob/master/vendor/github.com/openshift/library-go/pkg/operator/staticpod/controller/installer/manifests/installer-pod.yaml

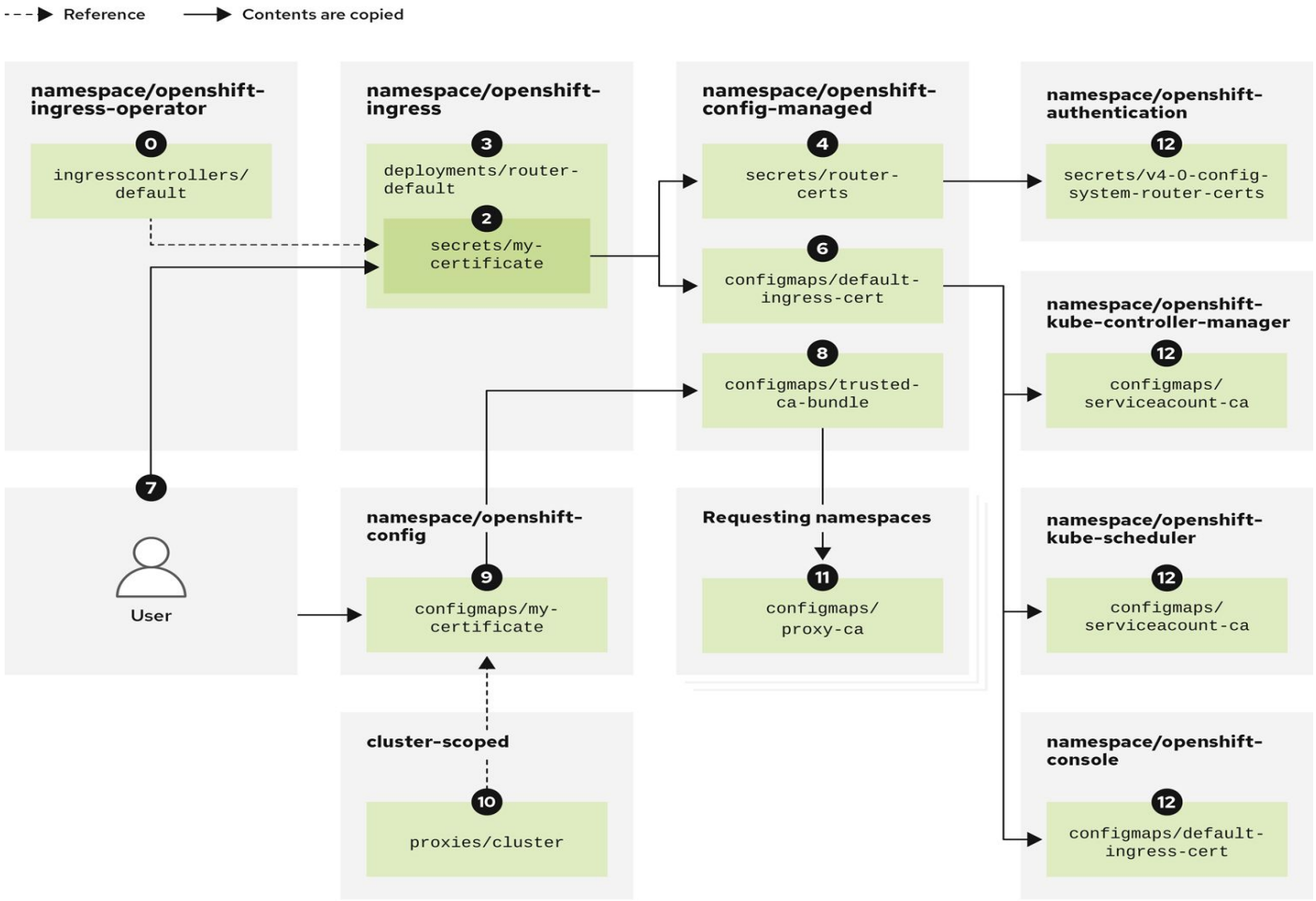
Ingress Controller Certificate

The Ingress Operator uses certificates for:

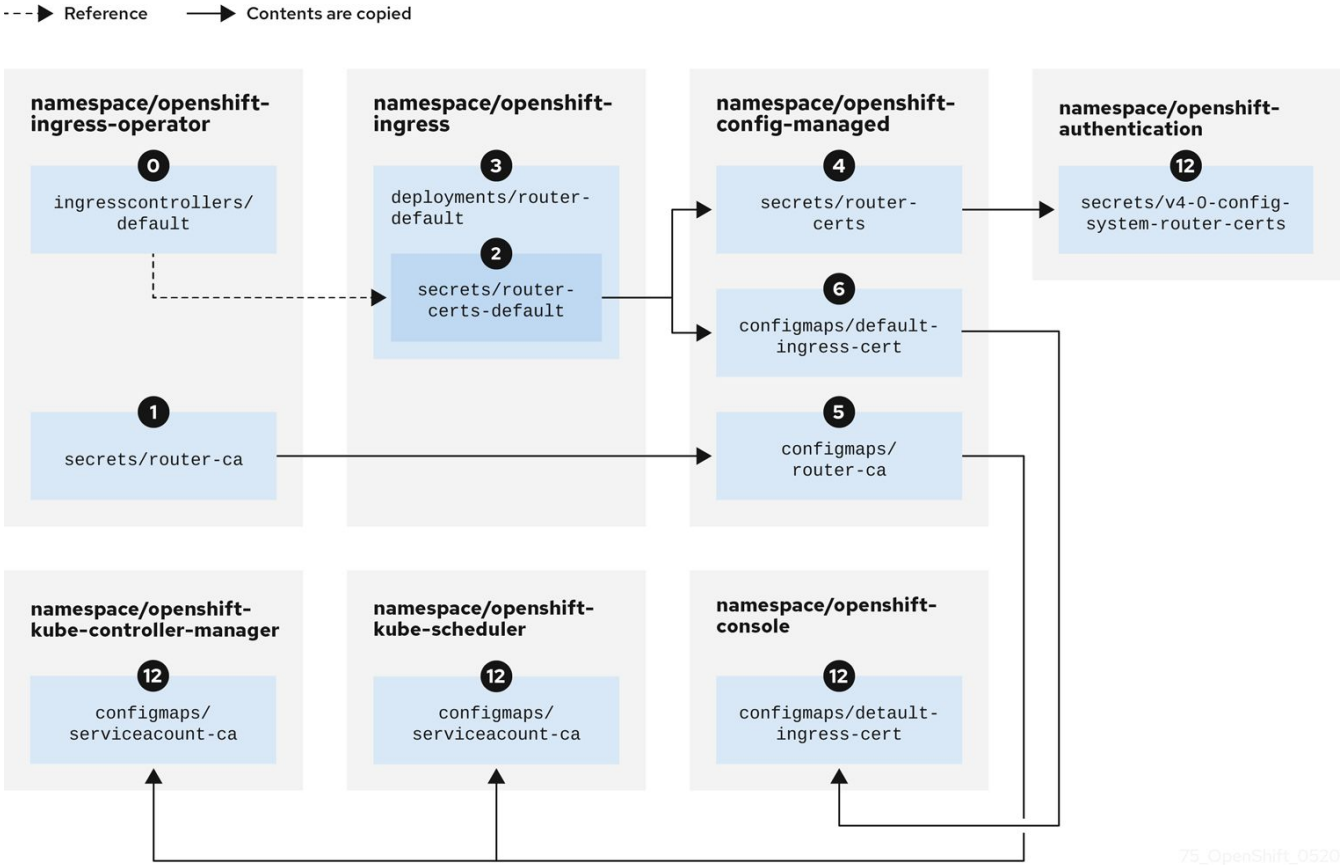
- Securing access to metrics for Prometheus.
- Securing access to routes.

To secure the access to Ingress Operator and Ingress Controller metrics, the Ingress Operator uses service serving certificates by requesting a certificate from the `service-ca` controller for its own metrics as well as for the Ingress Controllers. The certificate for metrics is present in a secret named `metrics-tls` in the `openshift-ingress-operator` namespace and `router-metrics-certs-<name>` in the `openshift-ingress` namespace.

Each Ingress Controller has a default certificate that it uses for secured routes that do not specify their own certificates. The wild-card certificate gets signed by the Ingress Operator CA certificate present inside the `openshift-ingress-operator` namespace as a secret named `router-ca`.



Custom certificate workflow



Default certificate workflow

1. An empty defaultCertificate field causes the Ingress Operator to use its self-signed CA to generate a serving certificate for the specified domain.
2. The default CA certificate and key generated by the Ingress Operator. Used to sign Operator-generated default serving certificates.
3. In the default workflow, the wildcard default serving certificate, created by the Ingress Operator and signed using the generated default CA certificate. In the custom workflow, this is the user-provided certificate.
4. The router deployment. Uses the certificate in secrets/router-certs-default as its default front-end server certificate.
5. In the default workflow, the contents of the wildcard default serving certificate (public and private parts) are copied here to enable OAuth integration. In the custom workflow, this is the user-provided certificate.
6. Transitional resource containing the certificate (public part) of the Operator-generated default CA certificate; read by OAuth and the web console to establish trust. This object will be removed in a future release.
7. The public (certificate) part of the default serving certificate. Replaces the configmaps/router-ca resource.
8. The user updates the cluster proxy configuration with the CA certificate that signed the ingresscontroller serving certificate. This enables components like auth, console, and the registry to trust the serving certificate.
9. The cluster-wide trusted CA bundle containing the combined Red Hat Enterprise Linux CoreOS (RHCOS) and user-provided CA bundles or an RHCOS-only bundle if a user bundle is not provided.
10. The custom CA certificate bundle, which instructs other components (for example, auth and console) to trust an ingresscontroller configured with a custom certificate.
11. The trustedCA field is used to reference the user-provided CA bundle.
12. The Cluster Network Operator injects the trusted CA bundle into the proxy-ca ConfigMap.
13. In OpenShift Container Platform 4.4, some components are transitioning from using router-ca to using default-ingress-cert.

Retrieve the Certificates from Secrets

- **Ingress Operator CA**

  $ oc get secret router-ca -o yaml -n openshift-ingress-operator

  $ oc get secret router-ca -n openshift-ingress-operator -o=custom-columns=":.data.tls\.crt" | tail -1 | base64 -d | openssl x509 -text -noout

- **Default Ingress Controller Certificate**

  $ oc get secret router-certs-default -o yaml -n openshift-ingress

  $ oc get secret router-certs-default -n openshift-ingress -o=custom-columns=":.data.tls\.crt" | tail -1 | base64 -d | openssl x509 -text -noout

- **Metrics TLS**

  $ oc get secret metrics-tls -n openshift-ingress-operator -o=custom-columns=":.data.tls\.crt" | tail -1 | base64 -d | openssl x509 -text -noout

  $ oc get secret router-metrics-certs-default -n openshift-ingress -o=custom-columns=":.data.tls\.crt" | tail -1 | base64 -d | openssl x509 -text -noout

Replace the Default Ingress Certificate

The certificate must have a `subjectAltName` extension of `*.apps.<clustername>.<domain>`.

- Create a ConfigMap that includes the certificate authority used to signed the new certificate:
  $ oc create configmap custom-ca --from-file=ca-bundle.crt=</path/to/example-ca.crt> -n openshift-config

- Update the cluster-wide proxy configuration with the newly created ConfigMap:
  $ oc patch proxy/cluster --type=merge --patch='{"spec":{"trustedCA":{"name":"custom-ca"}}}'

- Create a secret that contains the wildcard certificate and key:
  $ oc create secret tls <certificate> --cert=</path/to/cert.crt> --key=</path/to/cert.key> -n openshift-ingress

- Update the Ingress Controller configuration with the newly created secret:
  $ oc patch ingresscontroller.operator default --type=merge -p '{"spec":{"defaultCertificate": {"name": "<certificate>"}}}' -n openshift-ingress-operator

Renew the Ingress Certificate

- In case of custom certificate, the new certificate needs to be obtained with proper validity and then either the secret can be modified or re-created inside the `openshift-ingress` namespace.
- The default self-signed certificate can be easily renewed by just deleting the secret present inside `openshift-ingress` namespace and then deleting the router pods from the same namespace.
- Renewing or configuring the ingress certificate triggers the changes in secret and configmaps present inside `openshift-authentication`, `openshift-console` and `openshift-config-managed` namespace.

Useful GitHub Links for Ingress Controller

- certificate-publisher controller
  https://github.com/openshift/cluster-ingress-operator/blob/master/pkg/operator/controller/certificate-publisher/controller.go

- publish_certs.go
  https://github.com/openshift/cluster-ingress-operator/blob/master/pkg/operator/controller/certificate-publisher/publish_certs.go

- ca.go
  https://github.com/openshift/cluster-ingress-operator/blob/master/pkg/operator/controller/certificate/ca.go

- controller.go
  https://github.com/openshift/cluster-ingress-operator/blob/master/pkg/operator/controller/certificate/controller.go

Configuring the cluster-wide proxy during installation

- The proxy configuration needs to specified in the `install-config.yaml` file.

```
apiVersion: v1
baseDomain: my.domain.com
proxy:
  httpProxy: http://<username>:<pswd>@<ip>:<port>
  httpsProxy: http://<username>:<pswd>@<ip>:<port>
  noProxy: example.com
additionalTrustBundle: |
    -----BEGIN CERTIFICATE-----
    <MY_TRUSTED_CA_CERT>
    -----END CERTIFICATE-----
...
```

- The additionalTrustBundle generates a ConfigMap that is named `user-ca-bundle` in the `openshift-config` namespace that contains the specified CA certificate required for the proxying HTTPS connections. The Cluster Network Operator then creates a `trusted-ca-bundle` ConfigMap that merges these contents with the Red Hat Enterprise Linux CoreOS (RHCOS) trust bundle, and this ConfigMap is referenced in the Proxy object's `trustedCA` field.

The installation program does not support the proxy readinessEndpoints field.

Configuring the cluster-wide proxy after installation

- Create a ConfigMap inside `openshift-config` namespace with the appropriate custom CA for proxy.

```
apiVersion: v1
data:
  ca-bundle.crt: |
    <MY_PEM_ENCODED_CERTS>
kind: ConfigMap
metadata:
  name: user-ca-bundle
  namespace: openshift-config
```

- Edit the proxy to configure the necessary fields for the proxy using `oc edit proxy/cluster`.

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec:
  httpProxy: http://<username>:<pswd>@<ip>:<port>
  httpsProxy: http://<username>:<pswd>@<ip>:<port>
  noProxy: example.com
  readinessEndpoints:
  - http://www.google.com
  - https://www.google.com
  trustedCA:
    name: user-ca-bundle
```

Removing the cluster-wide proxy

- Use the oc edit command to modify the proxy.

  $ oc edit proxy/cluster

- Remove all spec fields from the Proxy object.

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
  name: cluster
spec: {}
status: {}
```

- The **trustedCA field can be left untouched under the spec if the custom CA certificates are getting used for some other components such as Ingress Controller.**

Quick tip

All the custom CA certificates needs to be added into a single ConfigMap for reference in proxy.

Known Proxy Issue

- There is an issue with the Machine Config Operator (MCO) supporting Day 2 proxy support, which describes when an existing non-proxied cluster is reconfigured to use a proxy. The MCO should apply newly configured proxy CA certificates in a ConfigMap to the RHCOS trust bundle; this is not working. As a workaround, manually add the proxy CA certificate to your trust bundle and then update the trust bundle:

```
$ cp /opt/registry/certs/<my_root_ca>.crt /etc/pki/ca-trust/source/anchors/
$ update-ca-trust extract
$ oc adm drain <node>
$ systemctl reboot
```

(BZ#1784201)

Service Serving Certificate Secrets

- The applications which requires TLS certificates out of the box for secure communication can use the service serving certificates. The TLS certificate generated by Service CA gets mounted at /var/serving-cert/ inside the pod with the name tls.crt and tls.key. So the application must be developed in such a way that it uses those certificates from the particular directory. The certificate and key are mounted from the secret which gets created by the Service CA operator.

- Service CA certificate which signs serving certificate can be found inside openshift-service-ca.
  $ oc get secret -n openshift-service-ca signing-key -o=custom-columns=":.data.tls\.crt" | tail -1 | base64 -d | openssl x509 -text -noout

- To secure communication to the service, generate a signed serving certificate and key pair into a secret in the same namespace as the service.
  $ oc annotate service <service-name> \
  service.beta.openshift.io/serving-cert-secret-name=<secret-name>

Add Service CA to pod's Truststore

- A Pod can access the service CA certificate by mounting a ConfigMap that is annotated with `service.beta.openshift.io/inject-cabundle=true`. Once annotated, the cluster automatically injects the service CA certificate into the `service-ca.crt` key on the ConfigMap. Access to this CA certificate allows TLS clients to verify connections to services using service serving certificates.

- An empty ConfigMap needs to be created with any name.
  $ oc create configmap <ConfigMap-name>

- Annotate the ConfigMap with `service.beta.openshift.io/inject-cabundle=true`:
  $ oc annotate configmap <configmap-name> \
  service.beta.openshift.io/inject-cabundle=true

- The ConfigMap needs to be mounted inside the pod by modifying the deployment/deployment configuration.

Add Custom CA to pod's Truststore

Troubleshooting

X509 certificate signed by unknown authority:

- This error comes up when a TLS certificate (server or client) is signed by an unknown CA (self-signed) and that unknown CA is not present in truststore to verify.

- The certificate served by a client or server including the CA certificate can be retrieved with the help of openssl command.
  $ openssl s_client -connect <route>:<port> -showcerts

- The certificate can be decoded as well by using the openssl command.
  $ openssl s_client -connect <route>:<port> -showcerts | openssl x509 -text -noout

- Any client or server certificate can be verified whether it is signed by a particula CA or not.
  $ openssl verify -CAfile <CA-cert> <server/client-cert>

Troubleshooting

Certificate is expired or is not yet valid:

- During the execution of oc commands, this error can come up or can be found in the API logs when the certificate of any of the component gets expired and it is trying to communicate with API.

- The validity of a TLS certificate can be checked with the help of these commands.
  $ openssl s_client -connect <route>:<port> -showcerts | openssl x509 -dates -noout
  $ openssl x509 -in <certificate> -dates -noout
  $ curl -kv https://<route>:<port>

- Sometimes this error comes up due to the inaccurate time of system. Maybe the certificate is having a proper validity but the system's time is configured wrong and the TLS certificate seems to be expired to it.

Troubleshooting

x509: certificate is valid for xxxxx not yyyy:

- Whenever a client sends a request to the server then the request carries the exact hostname which is specified to match with the CN (Common Name) or X509v3 SAN (Subject Alternative Name) present inside the server's certificate.

- This helps in SNI, to which backend pod or server the traffic needs to be send. This allows a server to present multiple certificates on the same IP address and TCP port number and hence allows multiple secure (HTTPS) websites (or any other service over TLS) to be served by the same IP address without requiring all those sites to use the same certificate.

- The CN and SAN for a certificate can be checked with openssl command and by default the ingress certificate and API certificate contains the SAN entry.
  $ openssl s_client -connect <API-URL>:6443 -showcerts | openssl x509 -text -noout

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.


linkedin.com/company/red-hat


youtube.com/user/RedHatVideos


facebook.com/redhatinc


twitter.com/RedHat

Red Hat