Procedure for updating Kubernetes cluster certificates

An Ezmeral Runtime Kubernetes cluster creates numerous SSL certificates for different purposes, such as securing the kube-apiserver https endpoint and authenticating admin kubectl requests. All of these certificates have a 1 year duration, and if they are allowed to expire, the cluster will be unusable until the certificates are renewed. New 1-year certificates are automatically generated whenever the cluster is upgraded, but if it's approaching one year since your cluster was created or upgraded, we strongly recommend checking the certificate expiry dates, as explained in Section 1, and, if necessary, following the steps in the remaining sections to prevent cluster downtime.

The certificates are generated from a root CA which is either created by Kubernetes itself (in which case its duration is 10 years) or supplied by the ECP admin at cluster creation time. Whatever the duration of the root CA, the duration of the certificates themselves is always 1 year.

Section 1: Check expiration dates of all relevant certs

1.1 Check the expiry of the core control plane certs on each Kubernetes master:

```
kubeadm certs check-expiration
```

(on Kubernetes versions prior to 1.20, use `kubeadm alpha certs check-expiration`)

If the kubeadm (alpha) certs command fails, you can check the expiry date of a specific certificate. The other cluster certificates will most likely all have the same expiry date. Of course, the main reason that the kubeadm command might fail is that all the certificates have already expired, but you can confirm it anyway:

```
cat /etc/kubernetes/pki/apiserver.crt |  openssl x509 -noout -enddate
```

1.2 To check kubelet client certs, run this on each Kubernetes node:

```
cat /var/lib/kubelet/pki/kubelet-client-current.pem |  openssl x509 -noout -
enddate
```

(note – for most ECP Kubernetes clusters, kubelet is configured to automatically rotate its certificates before they expire, and this will happen approx 1 month in advance of expiry. */var/lib/kubelet/kubeadm-flags.env* should include *RotateKubeletServerCertificate=true* and */var/lib/kubelet/config.yaml* should contain *rotateCertificates: true*.

1.3 For operator-related certs, run these commands on one of the master nodes (or wherever you have admin kubectl credetials):

```
# check the webhook configurations
kubectl get MutatingWebhookConfiguration hpecp-webhook -o jsonpath='{.webhooks[0].clientConfig.caBundle}' |
base64 -d | openssl x509 -noout -enddate
kubectl get MutatingWebhookConfiguration kubedirector-webhook -o jsonpath='{.webhooks[0].clientConfig.caBundle}'
| base64 -d | openssl x509 -noout -enddate

# also check the secrets used by the hpecp-agent and KD webhook services
kubectl -n hpecp get secret hpecp-validator-secret -o jsonpath='{.data.ca\.crt}' | base64 -d | openssl x509 -
noout -enddate
```

```
kubectl -n hpecp get secret hpecp-validator-secret -o jsonpath='{.data.app\.crt}' | base64 -d | openssl x509 -
noout -enddate
kubectl -n hpecp get secret kubedirector-validator-secret -o jsonpath='{.data.ca\.crt}' | base64 -d | openssl
x509 -noout -enddate
kubectl -n hpecp get secret kubedirector-validator-secret -o jsonpath='{.data.app\.crt}' | base64 -d | openssl
x509 -noout -enddate
```

1.4 On the primary controller, check the auth proxy cert for each K8S cluster, by cluster ID:

```
bdconfig --getk8sc
```

Then for each cluster ID:

```
cat /opt/bluedata/auth_proxy/<ID value here>/front-proxy-client-cert.pem |
openssl x509 -noout -enddate
```

Check the admin kubeconfig for a cluster. On a master node as the ECP install user, run:

```
cfg_filename=~/.kube/config # you can substitute any other kubeconfig file whose cert you want to check here
cat $cfg_filename | grep client-certificate-data | cut -d' ' -f 6 | base64 -d | openssl x509 -noout -enddate
```

Note that if running that last command on macOS, the base64 part will be "base64 -D" with a capital D.

Section 2. Renew all the Kubernetes cluster certificates.

2.1 As the ECP install user, on each Kubernetes master host in turn, run:

```
kubeadm certs renew all
```

(`kubeadm alpha certs renew all` on K8S versions < 1.20)

This command initially tries to collect cluster config using a kubectl command, which will of course fail if the certs have already expired. Therefore, the command might appear to hang for 30 seconds or so; but it will eventually renew all the cluster certs, even if they have already expired.

2.1.1 Restart all the kube-apiserver, kube-controller-manager, kube-scheduler and etcd pods in the kube-system namespace. Even if kubectl commands are still working, this should be done by deleting the underlying Docker containers on each master host in turn (instead of using *kubectl delete po*). This is because the pods mount a kubeconfig file (for example, /etc/Kubernetes/scheduler.conf for the kube-scheduler pods) from the underlying host, and those kubeconfig files are recreated as part of the cert renewal process; forcing the containers to be recreated ensures that the updated version of the relevant kubeconfig is loaded by the pod. Follow this process:

- On one of the master hosts, as root, run *docker ps | grep scheduler* to identify the two containers related to the kube-scheduler pod. Run *docker rm -f <id of container>* to forcibly delete each of those containers.
- Repeat for the kube-controller-manager pods (*docker ps | grep kube-controller-manager*)
- Wait a minute or so for new containers to be created (use *docker ps* again to check). If new containers do not start automatically in this time, try a *systemctl restart kubelet.*
- Repeat the above steps for remaining masters (completing each one before moving onto the next).

Now, run `watch kubectl get po -n kube-system` to wait for all the restarted pods to come to Ready state. It could take a couple of minutes for the kubectl commands to return anything at all. **Note:** If you're doing this process after your certificates have already expired, kubectl commands will still not be working at this point, because the certificates in kubeconfig have not yet been updated. In this case, run *docker ps* on the Kubernetes masters to see that the pods' containers have been newly created and are running steadily (not repeatedly restarting).
2.1.2 When the static pods have restarted, use kubeadm to check that the certificate expiry dates have been updated:

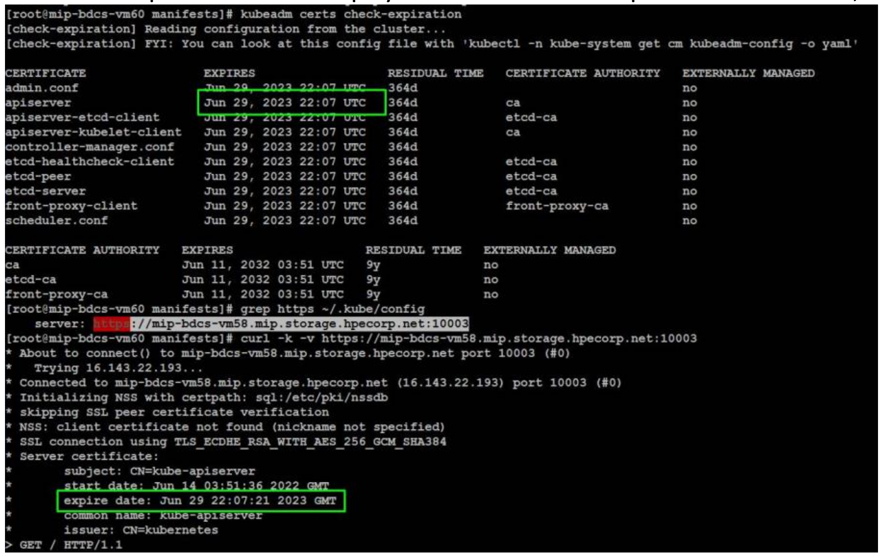
```
kubeadm (alpha) certs check-expiration
```

From a master host, find the apiserver gateway URL (*grep https ~/.kube/config*) and curl -k -v to that URL to check that expiration date matches the updated certificate expiries from kubeadm, in the format *curl –k –v https://<gateway fqdn>:<port number for api service>*

For example:

```
curl –k –v https:// mip-bd-vm157.mip.storage.hpecorp.net:10000
```

The curl output shows the expiry date on the kube-apiserver certificate, as shown in the screen shot below.



```
[root@mip-bdcs-vm60 manifests]# kubeadm certs check-expiration
[check-expiration] Reading configuration from the cluster...
[check-expiration] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'

CERTIFICATE                EXPIRES                  RESIDUAL TIME   CERTIFICATE AUTHORITY   EXTERNALLY MANAGED
admin.conf                 Jun 29, 2023 22:07 UTC   364d                                    no
apiserver                  Jun 29, 2023 22:07 UTC   364d            ca                      no
apiserver-etcd-client      Jun 29, 2023 22:07 UTC   364d            etcd-ca                 no
apiserver-kubelet-client   Jun 29, 2023 22:07 UTC   364d            ca                      no
controller-manager.conf    Jun 29, 2023 22:07 UTC   364d                                    no
etcd-healthcheck-client    Jun 29, 2023 22:07 UTC   364d            etcd-ca                 no
etcd-peer                  Jun 29, 2023 22:07 UTC   364d            etcd-ca                 no
etcd-server                Jun 29, 2023 22:07 UTC   364d            etcd-ca                 no
front-proxy-client         Jun 29, 2023 22:07 UTC   364d            front-proxy-ca          no
scheduler.conf             Jun 29, 2023 22:07 UTC   364d                                    no

CERTIFICATE AUTHORITY   EXPIRES                  RESIDUAL TIME   EXTERNALLY MANAGED
ca                      Jun 11, 2032 03:51 UTC   9y              no
etcd-ca                 Jun 11, 2032 03:51 UTC   9y              no
front-proxy-ca          Jun 11, 2032 03:51 UTC   9y              no
[root@mip-bdcs-vm60 manifests]# grep https ~/.kube/config
    server: https://mip-bdcs-vm58.mip.storage.hpecorp.net:10003
[root@mip-bdcs-vm60 manifests]# curl -k -v https://mip-bdcs-vm58.mip.storage.hpecorp.net:10003
* About to connect() to mip-bdcs-vm58.mip.storage.hpecorp.net port 10003 (#0)
*   Trying 16.143.22.193...
* Connected to mip-bdcs-vm58.mip.storage.hpecorp.net (16.143.22.193) port 10003 (#0)
* Initializing NSS with certpath: sql:/etc/pki/nssdb
* skipping SSL peer certificate verification
* NSS: client certificate not found (nickname not specified)
* SSL connection using TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
* Server certificate:
*       subject: CN=kube-apiserver
*       start date: Jun 14 03:51:36 2022 GMT
*       expire date: Jun 29 22:07:21 2023 GMT
*       common name: kube-apiserver
*       issuer: CN=kubernetes
> GET / HTTP/1.1
```

2.1.3 If you are performing these steps after the original certificates had already expired, the final part of this section is to check the logs of the kube-scheduler and kube-controller-manager pods on each host (which you forcibly recreated in section 2.1.1) to make sure they are up and running correctly with their regenerated kubeconfigs. If their logs are full of errors such as:

```
E0522 21:53:26.598561 1 leaderelection.go:224] error retrieving resource lock kube-system/kube-controller-manager: Unauthorized
E0522 21:54:37.749664 1 reflector.go:205] k8s.io/kubernetes/vendor/k8s.io/client-go/informers/factory.go:87: Failed to list *v1.PersistentVolume: Unauthorized
```

…it shows that they are still using expired certs. Re-visit the steps in section 2.1.1; you can also try rebooting each master host in turn, after first checking that the certs have been successfully recreated as shown in section 2.1.2.

2.2 Replace the kubeconfig file in the ECP install user's home directory on each master host with the updated version. Typically, the install user is root, so here we are replacing it into the root directory. But use the home directory of whatever user account you run kubectl commands as. To check who the ECP install user is, run *grep "^user" /etc/bluedata/bluedata.conf* on any of the ECP platform hosts.

```
cp /root/.kube/config /root/.kube/config.old
cp -i /etc/kubernetes/admin.conf /root/.kube/config
chown $(id -u):$(id -g) /root/.kube/config
chmod 777 /root/.kube/config
```

2.3 Kubelet client certificates will be automatically rotated approximately 1 month before they expire, on each host in the cluster. You don't need to do anything to update the kubelet config, but check it a week before expiry (`cat /var/lib/kubelet/pki/kubelet-client-current.pem |  openssl x509 -noout -enddate`) to make sure that the auto rotation happened. If you find any host whose kubelet certs were not rotated automatically, and the expiry date is less than 2 weeks from now, you can contact HPE support for assistance, or recreate them yourself:

2.3.1. Recreating kubelet client certs on masters.

```
systemctl stop kubelet
rm -rf /var/lib/kubelet/pki/ /etc/kubernetes/kubelet.conf
kubeadm init phase kubeconfig kubelet
kubeadm init phase kubelet-start
```

After running these steps, edit */etc/kubernetes/kubelet.conf* to replace the kube-apiserver endpoint with the gateway hostname and port number found in section 2.1.

2.3.2. Recreating kubelet client certs on workers.

First, run the kubeadm token create command on one of the masters, to determine the token string and the discovery-token-ca-cert-hash values:

```
kubeadm token create --print-join-command
```

Then run these commands on the worker whose kubelet client cert is about to expire. The format of the kubeadm join command is *kubeadm join phase kubelet-start <ECP gateway fqdn>:<port number of kube-apiserver endpoint on ECP gateway> --token <token obtained from token create command> --discovery-token-ca-cert-hash <hash value obtained from token create command>.* The gateway FQDN and port number are the same as the ones used in section 2.1 above.

```
systemctl stop kubelet
rm -rf /var/lib/kubelet/pki/ /etc/kubernetes/pki/ /etc/kubernetes/kubelet.conf
kubeadm join phase kubelet-start mip-bd-vm157.mip.storage.hpecorp.net:10000 --token
cs0etm.ua7fbmwuf1jz946l --discovery-token-ca-cert-hash
sha256:555f6ececd4721fed0269d27a5c7f1c6d7ef4614157a18e56ed9a1fd031a3ab8
```

Finally, check that the kubelet service is running fine on the worker whose cert was updated (systemctl status kubelet) – it should have automatically been restarted by the *kubeadm join* command - and check that the worker's status in the cluster is 'Ready' (*kubectl get nodes* from one of the master hosts). The additional steps of editing /etc/kubernetes/kubelet.conf, as performed on the masters in section 2.3.1, are not needed on workers.

2.4 Delete the mutating webooks, secrets and pods belonging to the hpecp-agent and kubedirector operators so that they get updated too. On the K8s master node (or wherever you have admin kubectl creds):

```
kubectl delete MutatingWebhookConfiguration hpecp-webhook
kubectl -n hpecp delete secret hpecp-validator-secret
kubectl -n hpecp delete pod -l name=hpecp-agent
kubectl delete MutatingWebhookConfiguration kubedirector-webhook
kubectl -n hpecp delete secret kubedirector-validator-secret
kubectl -n hpecp delete pod -l name=kubedirector
```

After the two operator pods are back in Ready state, you can re-check the expiration date on the new certs:

```
kubectl get MutatingWebhookConfiguration hpecp-webhook -o
jsonpath='{.webhooks[0].clientConfig.caBundle}' | base64 -d | openssl x509 -noout
-enddate
kubectl get MutatingWebhookConfiguration kubedirector-webhook -o
jsonpath='{.webhooks[0].clientConfig.caBundle}' | base64 -d | openssl x509 -noout
-enddate
kubectl -n hpecp get secret hpecp-validator-secret -o jsonpath='{.data.ca\.crt}'
| base64 -d | openssl x509 -noout -enddate
kubectl -n hpecp get secret hpecp-validator-secret -o jsonpath='{.data.app\.crt}'
| base64 -d | openssl x509 -noout -enddate
kubectl -n hpecp get secret kubedirector-validator-secret -o
jsonpath='{.data.ca\.crt}' | base64 -d | openssl x509 -noout -enddate
kubectl -n hpecp get secret kubedirector-validator-secret -o
jsonpath='{.data.app\.crt}' | base64 -d | openssl x509 -noout –enddate
```

Section 3. Updating the Kubernetes credentials used by the ECP primary controller

3.1 Check the ID of the Kubernetes cluster whose certs are being updated - this was obtained from the *bdconfig --getk8sc* command used earlier.

Then, as the ECP install user, attach to the Erlang console. ONLY quit the console with Ctrl+D, not Ctrl+C.

```
/opt/bluedata/common-install/bd_mgmt/bin/bd_mgmt attach
```

At the Erlang prompt, run the following commands (replacing the value of KubeClusterID with your own kubernetes cluster ID as found from bdconfig --getk8sc, and keeping the double quotes):

```
f().
KubeClusterID = "2",
{ok, Cluster} = bd_mgmt_db_k8s:get_cluster(KubeClusterID),
SomeMasterIP = hd(bd_mgmt_db_k8s:get_master_host_ips(KubeClusterID)).
{ok, AdminKubeConfig} = bd_mgmt_k8s_install:fetch_kubeadm_conf(SomeMasterIP),
bd_mgmt_db_k8s:update_cluster_kubeconfig(Cluster, AdminKubeConfig).
{ok, ProxyClientCerts} =
bd_mgmt_k8s_install:fetch_proxy_client_certs(SomeMasterIP),
bd_mgmt_db_k8s:update_cluster_client_certs(Cluster, ProxyClientCerts).
```

```
bd_kube_authn_proxy_starter:restart(KubeClusterID).
```

Now, hit Ctrl+D to exit the Erlang prompt, and run *docker ps* to check that the epic-authproxy container for the K8S cluster has recently been restarted.

Section 4: Verification

4.1 As a sanity test, go to the ECP UI, create a new K8S tenant, and create a new utility KD app inside there and make sure that the app comes to Configured state.