CDC Kafka certificate rotation @ CFA - Report

Summary

Not so long ago, on Sept 11, 2025 **CFA** had an incident in their **test environment**, where around ~6am their CDC Data Bridge listener **stopped** consuming data. We **identified** that the issue was related to **certificate** issues, but it was unclear exactly how/why. We saw in the [redacted] portal that the current certificate was issued in Aug 2025, despite the previous cert (received from CFA) was issued in May 18, and since our certificate rotation takes place once a year, it appeared that something forced a **premature** cert **rotation** in the **CFA** test cluster.

After asking CFA to fetch the latest certificates from the [redacted] portal and install them in their consumer, the problem was resolved.

Unfortunately there were crucial **questions** remained **unanswered**. **How** could the rotation happen **prematuraly**? Could this happen in **prod**? Which brought further questions and demand to review and **clarify** our **rotation policy** in our CDC Kafka based Data Bridge.

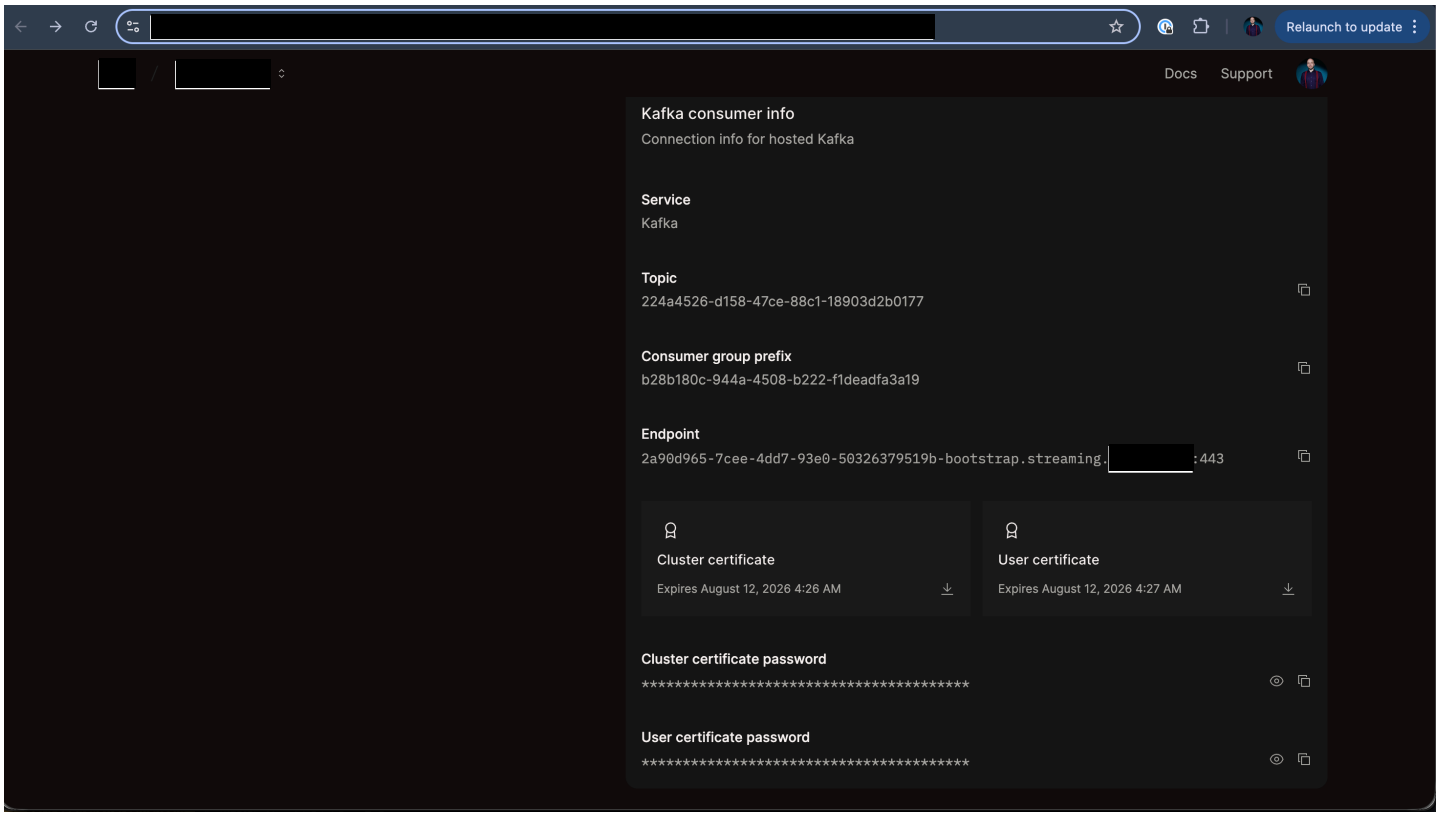
This **document** aims to **review** the **components** related to CDC certificates both [redacted] portal and Kubernetes, **clarifies** our rotation policies and tries to **answer** all questions of what really happened in us-east-1-prod.cfa-test-k8s.[redacted] on September 11.

Components

My first goal was to **identify** the **certificates** in question, and match them in **Kubernetes**. Then, I can review what's managing them and find the **reason** they were **rotated** prematuraly. There are many ways how we can tackle this, we could go by the [redacted] portal's source code, Strimzi's documentation or to find the **connection** between [redacted] **portal** and Kubernetes **manifests**. I chose the last one as we're going to look for **clues** in Kubernetes anyway, why not start the investigation by mapping it. Once I understand how everything is **connected**, I'll be able to answer any questions that come up.

[redacted] portal

For the CDC Data Bridge, we use **two certificates** with **two passwords**, one is the **cluster CA certificate** the second is the **user certificate**. The [redacted] **portal** is a great starting point to get all the **details** needed for the investigation:



Both of these **certificates** are **password** protected and used as an **authentication**, first for the cluster, then within the cluster, the customer authentication as well.

Both the **topic**, **consumer group prefix** and **endpoints** are important details to find our **target**.

Kubernetes

Since this is happening in the **test cluster**, we're working in **us-east-1-prod.cfa-test-k8s.[redacted]** here. The **ingress** is in the `cfa-test-1-lqt` namespace:

```
sendai@J9LXF7X3N2 sre-849 % kubectl get ingress  user-consumable-2a90d965-7cee-4dd7-93e0-
50326379519b -n cfa-test-1-lqt
NAME                                                    CLASS              HOSTS
                                                    ADDRESS
                                PORTS     AGE
user-consumable-2a90d965-7cee-4dd7-93e0-50326379519b   public-streaming   2a90d965-7cee-4dd7-
93e0-50326379519b-bootstrap.streaming.[redacted]    ab4b0b5381c044cdf9b9295896fadecd-
2015031346.us-east-1.elb.amazonaws.com   80, 443   718d
```

Config:

```
  - host: 2a90d965-7cee-4dd7-93e0-50326379519b-bootstrap.streaming.[redacted]
    http:
      paths:
      - backend:
          service:
            name: cfa-test-1-lqt-kafka-kafka-external-bootstrap
            port:
              number: 9094
        path: /
        pathType: Prefix
```

This ingress is **forwarding** connections to the `cfa-test-1-lqt-kafka-kafka-external-bootstrap` service, port **9094**:

```
sendai@J9LXF7X3N2 sre-849 % kubectl get service -n  cfa-test-1-lqt cfa-test-1-lqt-kafka-
kafka-external-bootstrap
NAME                                            TYPE        CLUSTER-IP      EXTERNAL-IP
PORT(S)    AGE
cfa-test-1-lqt-kafka-kafka-external-bootstrap   ClusterIP   100.78.171.76   <none>
 9094/TCP   718d
```

Which is using the following **pods** with **labelselectors**:

```
  ports:
  - name: tcp-external
    port: 9094
    protocol: TCP
    targetPort: 9094
  selector:
    strimzi.io/cluster: cfa-test-1-lqt-kafka
    strimzi.io/kind: Kafka
    strimzi.io/name: cfa-test-1-lqt-kafka-kafka
```

Which currently has **3 active pods**, the **3 brokers**:

```
sendai@J9LXF7X3N2 sre-849 % kubectl get endpoints -n  cfa-test-1-lqt cfa-test-1-lqt-kafka-
kafka-external-bootstrap
NAME                                             ENDPOINTS
               AGE
cfa-test-1-lqt-kafka-kafka-external-bootstrap
100.78.166.192:9094,100.78.167.132:9094,100.78.170.16:9094   718d
```

Here we go:

```
sendai@J9LXF7X3N2 sre-849 % kubectl get pods -n cfa-test-1-lqt -o wide | egrep "
(100.78.166.192|100.78.167.132|100.78.170.16)"
cfa-test-1-lqt-kafka-kafka-0                                      1/1     Running   0
48d   100.78.167.132   i-062d8e31db3e0b218   <none>           <none>
cfa-test-1-lqt-kafka-kafka-1                                      1/1     Running   0
48d   100.78.166.192   i-027a6fa3f35f90e64   <none>           <none>
cfa-test-1-lqt-kafka-kafka-2                                      1/1     Running   0
48d   100.78.170.16    i-0fa25bc0c39936875   <none>           <none>
```

Both the nginx and the brokers are **managed** by the **Kafka operator,** and so far these are all **described** in the **Kafka CRD**:

```
    - authentication:
        type: tls
      configuration:
        bootstrap:
          alternativeNames:
          - '*.streaming.[redacted]
          annotations: {}
          host: cfa-test-1-live-query-transport-bootstrap.streaming.[redacted]
        brokers:
        - broker: 0
          host: cfa-test-1-live-query-transport-broker-0.streaming.[redacted]
        - broker: 1
          host: cfa-test-1-live-query-transport-broker-1.streaming.[redacted]
        - broker: 2
          host: cfa-test-1-live-query-transport-broker-2.streaming.[redacted]
        class: public-streaming
```

Since our ingresses are NLBs with no SSL termination, these pods will be the **first** to do SSL handshaking, and the first step in authentication as well. Out of the two certs, these pods will **authenticate** with the **cluster certificate**.

If we look inside the pod, we'll see the certs **mounted** as tmpfs, indicating secrets/configmaps:

```
tmpfs           2.0G   12K  2.0G   1% /opt/kafka/cluster-ca-certs
tmpfs           2.0G   48K  2.0G   1% /opt/kafka/broker-certs
tmpfs           2.0G   12K  2.0G   1% /opt/kafka/client-ca-certs
```

Cluster certificate

From the pod's **config**, we can tell which **secrets** are mounted here:

```
  - name: cluster-ca
    secret:
      defaultMode: 292
      secretName: cfa-test-1-lqt-kafka-cluster-ca-cert
  - name: broker-certs
    secret:
      defaultMode: 292
      secretName: cfa-test-1-lqt-kafka-kafka-brokers
  - name: client-ca-cert
    secret:
      defaultMode: 292
      secretName: cfa-test-1-lqt-kafka-clients-ca-cert
```

Let's check the **cluster secret**:

```
apiVersion: v1
data:
  ca.crt:
    ...
  ca.p12:
    ...
 ownerReferences:
  - apiVersion: kafka.strimzi.io/v1beta2
    blockOwnerDeletion: false
    controller: false
    kind: Kafka
    name: cfa-test-1-lqt-kafka
    uid: 12b8d0c0-0921-41b0-b6a6-496416b422b7
```

I **assume**, we **found** our first certificate, the **cluster certificate**. Let's confirm. I saved the cluster cert from [redacted] portal as cluster.p12, and then we can compare the two payload's checksum:

```
sendai@J9LXF7X3N2 sre-849 % kubectl get secret -n cfa-test-1-lqt cfa-test-1-lqt-kafka-
cluster-ca-cert  -o jsonpath='{.data.ca\.p12}' | base64 -d | cksum
1357173798 1702
sendai@J9LXF7X3N2 sre-849 % cksum cluster.p12

1357173798 1702 cluster.p12
```

Confirmed, also, the ownerReferences tells us that it's **managed** by **Kafka**. Perfect, we **found** out cluster **CA certificate**!

An additional sanity **check**, the expiry date of the cluster CA:

```
sendai@J9LXF7X3N2 sre-849 % kubectl get secret -n cfa-test-1-lqt cfa-test-1-lqt-kafka-
cluster-ca-cert  -o jsonpath='{.data.ca\.crt}' | base64 -d | openssl x509 -noout -dates
notBefore=Aug 12 03:26:13 2025 GMT
notAfter=Aug 12 03:26:13 2026 GMT
```

User certificate

Now we know that unsurprisngly the **certificates** are stored in **secrets**, so we have to find **where** the **user certificate** is. For this, I chose a way simpler solution, just **reviewed** all the **secrets** in the namespace looking for the user secret's pattern. Finally, I **found** it in a secret **named** after the **consumer group** we saw on [redacted] **portal**:

```
sendai@J9LXF7X3N2 sre-849 % kubectl get secret -n cfa-test-1-lqt b28b180c-944a-4508-b222-
f1deadfa3a19-uc
NAME                                       TYPE     DATA   AGE
b28b180c-944a-4508-b222-f1deadfa3a19-uc    Opaque   5      152d
```

Let's **confirm** the same way as the cluster cert.

```
sendai@J9LXF7X3N2 sre-849 % kubectl get secret -n cfa-test-1-lqt b28b180c-944a-4508-b222-
f1deadfa3a19-uc -o jsonpath='{.data.user\.p12}' | base64 -d | cksum
487568583 3011
sendai@J9LXF7X3N2 sre-849 % cksum user.p12

487568583 3011 user.p12
```

As you can see, we **found** the user cert as well, and as we check its **ownerReferences**:

```
ownerReferences:
 - apiVersion: kafka.strimzi.io/v1beta2
   blockOwnerDeletion: false
   controller: false
   kind: KafkaUser
   name: b28b180c-944a-4508-b222-f1deadfa3a19-uc
   uid: d1b4137e-0b99-4f18-bd25-0f50d725fc10
```

It's owned by the **KafkaUser CRD** with the **consumer group name**.

Great, now we **know** where the **certs** are coming from, we have the CLI tools to **confirm** their expiry date, we can move on to the **rotation policy**.

Certificate rotation

We have two certs, the cluster CA cert and the user cert and trying to find the **rotation policies**, relevant **log** lines. For this, I had to track down exactly which **components** are **responsible** for the cert **rotations**. The cluster CA cert is rotated by the cluster operator, and the user certs are rotated by the user operator. It's important that when the cluster CA cert is rotated, that also brings new user certs as well. Configurable maintenance windows can affect the certificate rotation time, but we're not using them in our deployments.

There are **two rotation types** in Kafka operator RENEW_CERT and REPLACE_KEY. By default, **RENEW_CERT** is active, which means that **private key** is left **untouched**, and only the **public key** (**certificate**) is **rotated**. This is crucial for the 1 month renewal window to both old and new certificates to work. REPLACE_KEY mode would rotate the private key as well, but this option is reserved for forced manual control via the force-replace annotation. This logic is described here.

It is crucial to declare that certificate **rotation can happen outside** of the **standard** rotation **logic**. These are edge cases, but the operator code contains branches where this happens, for example if the operator believes that the cert is incomplete/corrupted.

Important to understand though that the **certificate rotation** only affects the **public key**. Even **if** there's a **premature rotation**, because of a potential corrupted certificate, the current certificate fetched earlier and in use in our consumers **will** still **work** until it expires. The rotation **doesn't invalidate** previously released **certificates**, so it's a low risk event. Only the **expiration** date, what **matters**.

We can **identify** renewal **events** via **logs**, searching for "Renewal" in the **kafka-operator namespace**. Unfortunately there is **no metric** in the kafka-operator **related** to **cert renewals**, so we have to work with log events.

Cluster CA cert

Both the cluster CA cert and user cert are, by **default valid** for 1 year (365 days), and **rotation** takes place by **default 30 days** before the expiration date. In that **last 30 days**, both the **old** and the **new** certs are **working**, so our clients have **30 days window** to **update** their keys. This is achieved by rotating only the public key, the private key remains, so both the old and new public keys will work with the constant private key. The old key will stop working only because of its expiration date.

Both the default certificate **validity length** and renewal days are **configurable**, both for the cluster CA and the user certs, but from the Kafka CRD we can only change the cluster CA parameters:

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
# ...
spec:
# ...
  clusterCa:
    renewalDays: 30
    validityDays: 365
    generateCertificateAuthority: true
```

User certs

The user certs **expiration** policy can be **configured** via the clientsCa parameters in the Kafka CRD:

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
# ...
spec:
# ...
  clientsCa:
    renewalDays: 30
    validityDays: 365
    generateCertificateAuthority: true
# ...
```

And they are translated into **environment** variables in the **entity operator**:

```
    - name: STRIMZI_CA_VALIDITY
      value: "365"
    - name: STRIMZI_CA_RENEWAL
      value: "30"
```

By default, the **user operator** is part of the **entity operator**, which is managed by the Kafka operator:

```
cfa-test-1-lqt-kafka-entity-operator                          1/1     1            1
715d
```

We're using the default, **entity embedded user operator**, but Kafka operator also supports to have it deployed separately.

Cert management

So far we understand **where** the Kafka **certs** are in **Kubernetes**, what is the **default rotation policy (**we use the default), how can we **reconfigure** the rotation policy parameters (**validity** day, **renewal** day), the role of cluster operator and user operator. I also want to add that Kafka operator supports forcing a certificate rotation, by the `strimzi.io/force-renew` annotation (this is using the **RENEW_CERT** option mentioned before):

```
kubectl annotate secret my-cluster-cluster-ca-cert -n my-project strimzi.io/force-
renew="true"
```

Kafka operator also support using our own key, instead of using a managed key. The cert **rotations** are also **logged**, so we can find them in the logs as well, in the `kafka-operator` namespace:

```
2025-08-12T04:26:13+01:00 {} 2025-08-12 03:26:13 INFO  Ca:1178 - Reconciliation #41138(timer)
Kafka(cfa-test-1-lqt/cfa-test-1-lqt-kafka): Renewing CA with
subject=Subject(organizationName='io.strimzi', commonName='clients-ca v0', dnsNames=[],
ipAddresses=[])
```

Putting it all together

We're in the **finish line** to unwrap **what happened** here. Quick recap: we've discussed **how** to **connect** the **cert** on the [redacted] **portal** to **Kubernetes** objects, how to **verify** expiry dates and renewal days, **what** components to **monitor**, what **log entries** to look for. We also **identified** code path in the Kafka operator where the key rotation is happening **outside** of the standard rotation policy, and discussed why it's a low risk event as it **wouldn't invalidate** previously released certificates.

We know that the incident **started** to happen on **September 11, 2025**, **~6am EST**. We also found **log** entries, indicating that Kafka cluster **indeed rotated** the keys **30 days before**:

```
2025-08-12 03:26:13+01:00 {} 2025-08-12 03:26:13 INFO  Ca:1178 - Reconciliation #41138(timer)
Kafka(cfa-test-1-lqt/cfa-test-1-lqt-kafka): Renewing CA with
subject=Subject(organizationName='io.strimzi', commonName='clients-ca v0', dnsNames=[],
ipAddresses=[])
```

So far these are all **normal events**. We also know by the **cert** that we received from CFA the certificate that **was rotated** in September had the **expiration** date of **May 18, 2026**:

```
issuer=O=io.strimzi, CN=cluster-ca v0
subject=O=io.strimzi, CN=cluster-ca v0
notBefore=May 18 19:25:46 2025 GMT
notAfter=May 18 19:25:46 2026 GMT
```

The question is.. **why** our **Kafka rotated** the certificate **before** the expiration date of **May 18, 19:25:46 2026** (-30 days, so ~April 19, 2026)?

The **answer** is, **it didn't**.

Here are the **renewal dates** of `cfa-test-1-lqt/cfa-test-1-lqt-kafka` since its **inception**:

```
2024-09-11T04:24:20+01:00 {} 2024-09-11 03:24:20 INFO  Ca:1178 - Reconciliation
#187571(timer) Kafka(cfa-test-1-lqt/cfa-test-1-lqt-kafka): Renewing CA with
subject=Subject(organizationName='io.strimzi', commonName='clients-ca v0', dnsNames=[],
ipAddresses=[])
2025-08-12T04:26:13+01:00 {} 2025-08-12 03:26:13 INFO  Ca:1178 - Reconciliation #41138(timer)
Kafka(cfa-test-1-lqt/cfa-test-1-lqt-kafka): Renewing CA with
subject=Subject(organizationName='io.strimzi', commonName='clients-ca v0', dnsNames=[],
ipAddresses=[])
```

Exactly as expected, the **previous rotation** before **August 12, 2025** was, **~11 months** (365 - 30 days) **ago** on September 11, 2024. To **confirm** that the timestamps are **exactly** what we assume, let's check with Python:

```
>>> str(datetime.datetime.fromisoformat('2025-08-12T04:26:13+01:00') -
datetime.timedelta(days=365-30))
'2024-09-11 04:26:13+01:00'
```

As you can see, the **previous orderly renewal** was exactly as the **log shows** to be (+-~2 minutes). Before this, there are no more renewal logs, but if we check one year before, 2024:

```
>>> str(datetime.datetime.fromisoformat('2024-09-11T04:24:20+01:00') -
datetime.timedelta(days=365-30))
'2023-10-12 04:24:20+01:00'
```

We're going to **find** the **initialization** of the Kafka cluster:

```
2023-10-12T04:23:26+01:00 {filename="/var/log/pods/kafka-operator_strimzi-cluster-operator-
675d86f8c-hqt2s_421633db-2d54-4c79-8151-ca8ea3a21364/strimzi-cluster-operator/0.log",
node_name="i-02f4c959d34e1ddb1", pod="strimzi-cluster-operator-675d86f8c-hqt2s",
stream="stdout"} 2023-10-12 03:23:26 INFO  OperatorWatcher:38 - Reconciliation #491(watch)
Kafka(cfa-test-1-lqt/cfa-test-1-lqt-kafka): Kafka cfa-test-1-lqt-kafka in namespace cfa-
test-1-lqt was ADDED
```

which is further **confirmed** by the Kafka CRD's **creationTimestamp**:

```
creationTimestamp: "2023-10-12T03:23:26Z"
```

This **proves** that the Kafka in the **cfa-test-1-lqt** namespace in the **us-east-1-prod.cfa-test-k8s.[redacted] cluster** behaves **correctly**, **rotates** certificates every **(365 - 30) days**, which is ~11 months, **leaving** the customer to update its **certificates 1 month window**, until the the 365 days validity certificate expires.

Ok, so we **confirmed** that the **Kafka cluster didn't rotate** its certificates **out of order**. The question remains, **why** we saw the **certificate** being **created** in **May 18, 2025**, with the expiration date May 18, 2026?

If you scroll back to the **beginning**, when we saw the rotation happen on August 11, 2025, we **assumed** it was a **prematurely** rotating certificate, as the **cert which was replaced** had these creation and expiration dates:

```
issuer=O=io.strimzi, CN=cluster-ca v0
subject=O=io.strimzi, CN=cluster-ca v0
notBefore=May 18 19:25:46 2025 GMT
notAfter=May 18 19:25:46 2026 GMT
```

We were able to **confirm** that the Kafka in the **test server didn't rotate** during this time. So **where** these **timestamps** are **coming from**?

The **answer** is:

```
sendai@J9LXF7X3N2 cert_from_cfa % kubectl get secret -n cfa-prod-1-lqt cfa-prod-1-lqt-kafka-cluster-ca-cert -o jsonpath='{.data.ca\.crt}' | base64 -d | openssl x509 -noout -startdate -enddate
notBefore=May 18 19:25:46 2025 GMT
notAfter=May 18 19:25:46 2026 GMT
```

It is **coming** from the **production CFA cluster**. Unfortunately the **certificate** we **received** from CFA was **accidentally mixed up** and we received the **production cluster cert**, instead of the **test cluster cert**. So far, this is only a theory but it's a pretty good one, the **timestamps match**, down to the **second**.

Let's further **confirm** this, by getting the renewal **event timestamps** from production, **us-east-1-prod.cfa-prod-k8s.[redacted]**

```
2024-06-17T20:24:53+01:00 {} 2024-06-17 19:24:53 INFO  Ca:1178 - Reconciliation #80222(timer) Kafka(cfa-prod-1-lqt/cfa-prod-1-lqt-kafka): Renewing CA with subject=Subject(organizationName='io.strimzi', commonName='clients-ca v0', dnsNames=[], ipAddresses=[])
2025-05-18T20:25:46+01:00 {} 2025-05-18 19:25:46 INFO  Ca:1178 - Reconciliation #114158(timer) Kafka(cfa-prod-1-lqt/cfa-prod-1-lqt-kafka): Renewing CA with subject=Subject(organizationName='io.strimzi', commonName='clients-ca v0', dnsNames=[], ipAddresses=[])
```

These are the two **renewal events** for **production**, one for **2024** and for **2025**, and as you can see, the **timestamp** in 2025 **matches** exactly the **cert creation** timestamp we **received**, **May 18, 19:25:46, 2025**.

At this stage we proved **99.9%** that the **cert** we **received** was from the **production cluster,** not from the test. The **final step** of the **confirmation** is to **compare** the cert itself, via the **checksum** of the **production cluster CA cert** and the one we **received**:

```
sendai@J9LXF7X3N2 cert_from_cfa % kubectl get secret -n cfa-prod-1-lqt cfa-prod-1-lqt-kafka-cluster-ca-cert -o jsonpath='{.data.ca\.crt}' | base64 -d | cksum
1658897113 1854
sendai@J9LXF7X3N2 cert_from_cfa % cksum cluster.crt
1658897113 1854 cluster.crt
```

As you can see, this **proves** our assumption **unequivocally**. Accidentally we **received** the **production** cluster CA cert instead of the test cluster CA cert, which led us to incorrectly **assume** that the certificate was still **valid until May 18, 2026**, when in reality it was **up to rotation** at a **precise** and **accurate timing**, completely **in order**.

Commands summary

Query the **cluster CA cert**'s dates:

```
kubectl get secret -n <namespace> <kafka name>-kafka-cluster-ca-cert  -o jsonpath='{.data.ca\.crt}' | base64 -d | openssl x509 -noout -dates
```

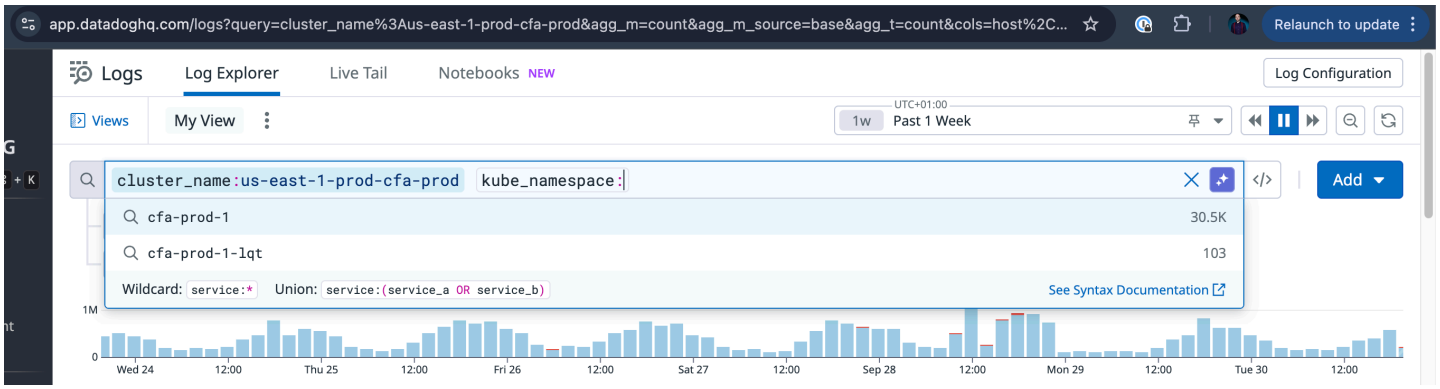
Query the **user cert**'s dates:

```
kubectl get secret -n <namespace> <consumer group prefix from [redacted] portal> -o jsonpath='{.data.user\.crt}' | base64 -d | openssl x509 -noout -dates
```

Log filter for renewing **event** in Loki:

```
{namespace="kafka-operator"} |~ "Renewing"
```

Unfortunately, it appears we **don't collect logs** for **kafka-operator** namespace in **DataDog**, this should be corrected:



I'd **recommend** configuring **mail** sending **trigger** for this **pattern** to our clients as the **notification** of the beginning of the **renewal window**.

Conclusion

We had an **incident** on September 11, ~6am EST as the **CFA CDC Listener stopped consuming** messages from Kafka in the **us-east-1-prod.cfa-test-k8s.[redacted] cluster**. We **identified** the reason as a Kafka **certificate expiration** and **rotation**, that took place on August 11, 2025. We asked CFA to **update** the **certs** to the **latest** via the [redacted] **portal**, at the same time **send** us their **current cert**, so we can **confirm** if the **rotation** was **unexpected** or **in order**. **Accidentally** we received the cluster certificate **from** the **production CFA server**, which has the creation timestamp of **May 18, 2025** and expiration time of May 18, 2026 which **misled** us to **believe** that our Kafka **rotated** its certificate **prematurely**.

The truth is, **logs confirm** that the **CFA test cluster Kafka rotated correctly** (365 - 30) days and **precisely**, every time since its creation. **Logs** and **binary diff** also **confirms** that we **received** the **cluster CA** from the **production cluster**.

Key takeaways

- **Kafka operator** in CFA test cluster **rotated** the certificates **correctly**, **on time**
- **Premature rotation** is a reality for edge cases, but that **wouldn't invalidate** previously released **certificates**
- We should **configure** a **mail** sending trigger for the **renewal** string from logs, **notifying** our **clients** the start of their 30 days renewal window
- DataDog currently **doesn't** pick up the **kafka-operator** namespace **logs** today, we should configure it

sendai @[redacted] 2025