Juniper Networks | Engineering Simplicity

Streaming API Guide

Published
2024-03-21

RELEASE
4.4

Table of Contents

Introduction

This guide describes how to extract data from Paragon Active Assurance via the product's streaming API.

The API as well as the streaming client are included in the Paragon Active Assurance installation. However, a bit of configuration is needed before you can use the API. This is covered in the "Configuring the Streaming API" on page 1 chapter.

Configuring the Streaming API

IN THIS SECTION

Overview

This chapter describes how to configure the Streaming API to allow subscribing to metrics messages via Kafka.

pr

Below we will go through:

- How to enable the Streaming API
- How to configure Kafka to listen to external clients

- How to configure Kafka to use ACLs and set up SSL encryption for said clients

What Is Kafka?

Kafka is an event-streaming platform that allows real-time capture of data sent from various event sources (sensors, databases, mobile devices) in the form of event streams, as well as durable storing of these event streams for later retrieval and manipulation.

With Kafka it is possible to manage the event streaming end-to-end in a distributed, highly scalable, elastic, fault-tolerant, and secure manner.

> **NOTE**: Kafka can be configured in many different ways and was designed for scalability and redundant systems. This document focuses only on how to configure it to make use of the Streaming API feature found in Paragon Active Assurance Control Center. For more advanced setups we refer to the official Kafka documentation: kafka.apache.org/26/documentation.html.

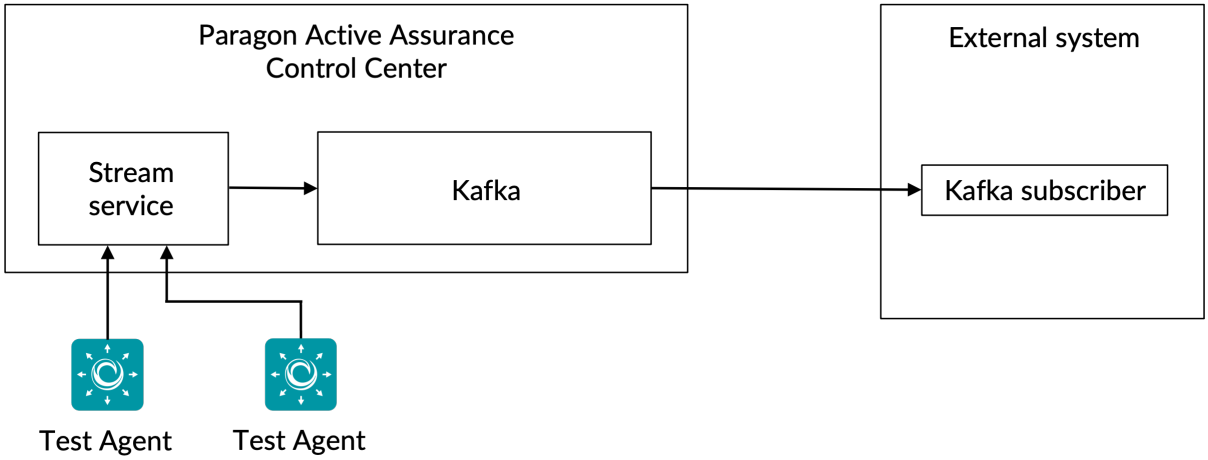
Terminology

- **Kafka:** Event-streaming platform.
- **Kafka topic:** Collection of events.
- **Kafka subscriber/consumer:** Component responsible for retrieval of events stored in a Kafka topic.
- **Kafka broker:** Storage layer server of a Kafka cluster.
- **SSL/TLS:** SSL is a secure protocol developed for sending information securely over the Internet. TLS is the successor of SSL, introduced in 1999.
- **SASL:** Framework that provides mechanisms for user authentication, data integrity checking, and encryption.
- **Streaming API subscriber:** Component responsible for retrieval of events stored in topics defined in Paragon Active Assurance and meant for external access.
- **Certificate Authority:** A trusted entity that issues and revokes public key certificates.
- **Certificate Authority root certificate:** Public key certificate that identifies a Certificate Authority.

How the Streaming API Works

As previously mentioned, the Streaming API allows external clients to retrieve information about metrics from Kafka.

All metrics collected by the Test Agents during a test or monitoring task are sent to the Stream service. After a processing phase, the Stream service publishes those metrics on Kafka together with additional metadata.



Kafka Topics

Kafka has the concept of *topics* to which all data is published. In Paragon Active Assurance there are many such Kafka topics available; however, only a subset of these are meant for external access.

Each Paragon Active Assurance account in Control Center has two dedicated topics. Below, `ACCOUNT` is the account short name:

- `paa.public.accounts.{ACCOUNT}.metrics`
  - All metrics messages for the given account is published to this topic
  - Large amounts of data
  - High update frequency
- `paa.public.accounts.{ACCOUNT}.metadata`
  - Contains metadata related to the metrics data, for example the test, monitor or Test Agent associated with the metrics
  - Small amounts of data
  - Low update frequency

Enabling the Streaming API

NOTE: These instructions are to be run on the Control Center server using `sudo`.

Since the Streaming API adds some overhead to the Control Center, it is not enabled by default. To enable the API, we must first enable publishing of metrics to Kafka in the main configuration file:

- `/etc/netrounds/netrounds.conf`

```
KAFKA_METRICS_ENABLED = True
```

WARNING: Enabling this feature might impact Control Center performance. Make sure you have dimensioned your instance accordingly.

Next, to enable forwarding of these metrics to the correct Kafka topics:

- `/etc/netrounds/metrics.yaml`

```
streaming-api: true
```

To enable and start the Streaming API services, run:

```
sudo ncc services enable timescaledb metrics
sudo ncc services start timescaledb metrics
```

Finally, restart the services:

```
sudo ncc services restart
```

Verifying That the Streaming API Works in Control Center

NOTE: These instructions are to be run on the Control Center server.

You can now verify that you are receiving metrics on the correct Kafka topics. To do so, install the `kafkacat` utility:

```
sudo apt-get update
sudo apt-get install kafkacat
```

If you have a test or monitor running in Control Center, you should be able to use `kafkacat` to receive metrics and metadata on these topics.

Replace `myaccount` with the short name of your account (this is what you see in your Control Center URL):

```
export METRICS_TOPIC=paa.public.accounts.myaccount.metrics
export METADATA_TOPIC=paa.public.accounts.myaccount.metadata
```

You should now see metrics by running this command:

```
kafkacat -b ${KAFKA_FQDN}:9092 -t ${METRICS_TOPIC} -C -e
```

To view metadata, run the following command (note that this will not update as frequently):

```
kafkacat -b ${KAFKA_FQDN}:9092 -t ${METADATA_TOPIC} -C -e
```

NOTE:
`kafkacat`"Client Examples" on page 14

This verifies that we have a working Streaming API from within Control Center. However, most likely you are interested in accessing the data from an external client instead. The next section describes how to open up Kafka for external access.

Opening Up Kafka for External Hosts

NOTE: These instructions are to be run on the Control Center server.

By default Kafka running on the Control Center is configured to only listen on localhost for internal use. It is possible to open up Kafka for external clients by modifying Kafka settings.

Connecting to Kafka: Caveats

CAUTION: Please read this carefully, since it is easy to run into connection issues with Kafka if you have not understood these concepts.

In the Control Center setup described in this document, there is only a single Kafka broker.

However, note that a Kafka broker is meant to run as part of a Kafka *cluster* which may consist of many Kafka brokers.

When connecting to a Kafka broker, an initial connection is set up by the Kafka client. Over this connection the Kafka broker in turn will return a list of "advertised listeners", which is a list of one or more Kafka brokers.

On receiving this list, the Kafka client will disconnect, then reconnect to one of these advertised listeners. The advertised listeners must contain hostnames or IP addresses that are accessible to the Kafka client, or the client will fail to connect.

If SSL encryption is used, involving an SSL certificate which is tied to a particular hostname, it is even more important that the Kafka client receives the correct address to connect to, since otherwise the connection may be rejected.

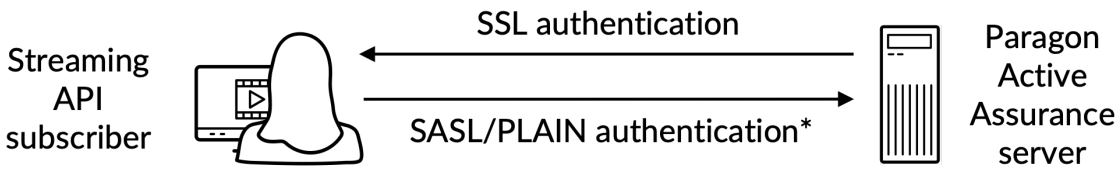
Read more about Kafka listeners here: www.confluent.io/blog/kafka-listeners-explained

SSL/TLS Encryption

To make sure only trusted clients are allowed to access Kafka and the Streaming API, we must configure the following:

- *Authentication:* Clients must provide username and password through an SSL/TLS secure connection between the client and Kafka.
- *Authorization:* Authenticated clients can perform tasks regulated by ACLs.

Here is an overview:



*) Username/password authentication performed on an SSL-encrypted channel

To fully understand how the SSL/TLS encryption works for Kafka, please refer to the official documentation: docs.confluent.io/platform/current/kafka/encryption.html

SSL/TLS Certificate Overview

NOTE: In this subsection we will use the following terminology:

Certificate: An SSL certificate signed by a Certificate Authority (CA). Each Kafka broker has one.

Keystore: The keystore file that stores the certificate. The keystore file contains the private key of the certificate; therefore, it needs to be kept safely.

Truststore: A file containing the trusted CA certificates.

To set up the authentication between an external client and Kafka running in Control Center, both sides must have a *keystore* defined with a related certificate signed by a Certificate Authority (CA) together with the CA root certificate.

In addition to this, the client must also have a *truststore* with the CA root certificate.

The CA root certificate is common to the Kafka broker and Kafka client.

Creating the Required Certificates

This is covered in the "Appendix" on page 17 .

Kafka Broker SSL/TLS Configuration in Control Center

NOTE: These instructions are to be run on the Control Center server.

NOTE: Before continuing, you must create the keystore which contains the SSL certificate by following the instructions in the "Appendix" on page 17 . The paths mentioned below come from these instructions.

The SSL keystore is a file stored on disk with the file extension .jks.

Once you have the required certificates created for both the Kafka broker and the Kafka client available, you can continue by configuring the Kafka broker running in Control Center. You need to know the following:

- `<public_hostname>`: The public hostname of Control Center; this must be resolvable and accessible by Kafka clients.
- `<keystore_pwd>`: The keystore password provided when creating the SSL certificate.
- `<admin_pwd>` and `<client_pwd>`: These are the passwords you want to set for the admin and client user respectively. Note that you can add more users, as indicated in the example.

Edit or append (with `sudo` access) the properties below in `/etc/kafka/server.properties`, inserting the above variables as shown:

WARNING: Do not remove `PLAINTEXT://localhost:9092`; this will break Control Center functionality since internal services will not be able to communicate.

```
...
# The addresses that the Kafka broker listens on.
listeners=PLAINTEXT://localhost:9092,SASL_SSL://0.0.0.0:9093

# These are the hosts advertised back to any client connecting.
advertised.listeners=PLAINTEXT://localhost:9092,SASL_SSL://<public_hostname>:9093
...
####### CUSTOM CONFIG
# SSL CONFIGURATION
ssl.endpoint.identification.algorithm=
ssl.keystore.location=/var/ssl/private/kafka.server.keystore.jks
ssl.keystore.password=<keystore_pwd>
ssl.key.password=<keystore_pwd>
ssl.client.auth=none
ssl.protocol=TLSv1.2

# SASL configuration
```

```
sasl.enabled.mechanisms=PLAIN
listener.name.sasl_ssl.plain.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginMo
dule required \
    username="admin" \
    password="<admin_pwd>" \
    user_admin="<admin_pwd>" \
    user_client="<client_pwd>";
# NOTE more users can be added with user_<name>=

# Authorization, turn on ACLs
authorizer.class.name=kafka.security.authorizer.AclAuthorizer
super.users=User:admin
```

Setting up Access Control Lists (ACLs)

Turning On ACLs on localhost

WARNING: We must first set up ACLs for localhost, so that Control Center itself can still access Kafka. If this is not done, things will break.

```
######### ACLs entries for anonymous users
/usr/lib/kafka/bin/kafka-acls.sh \
   --authorizer kafka.security.authorizer.AclAuthorizer \
   --authorizer-properties zookeeper.connect=localhost:2181 \
   --add --allow-principal User:ANONYMOUS --allow-host 127.0.0.1 --cluster

/usr/lib/kafka/bin/kafka-acls.sh \
   --authorizer kafka.security.authorizer.AclAuthorizer \
   --authorizer-properties zookeeper.connect=localhost:2181 \
   --add --allow-principal User:ANONYMOUS --allow-host 127.0.0.1 --topic '*'

/usr/lib/kafka/bin/kafka-acls.sh \
   --authorizer kafka.security.authorizer.AclAuthorizer \
   --authorizer-properties zookeeper.connect=localhost:2181 \
   --add --allow-principal User:ANONYMOUS --allow-host 127.0.0.1 --group '*'
```

We then need to enable ACLs for external read-only access, so that external users are allowed to read the `paa.public.*` topics.

NOTE: For more fine-grained control, please refer to the official Kafka documentation.

```
######### ACLs entries for external users
/usr/lib/kafka/bin/kafka-acls.sh \
   --authorizer kafka.security.authorizer.AclAuthorizer \
   --authorizer-properties zookeeper.connect=localhost:2181 \
   --add --allow-principal User:* --operation read --operation describe \
   --group 'NCC'

/usr/lib/kafka/bin/kafka-acls.sh \
   --authorizer kafka.security.authorizer.AclAuthorizer \
   --authorizer-properties zookeeper.connect=localhost:2181 \
   --add --allow-principal User:* --operation read --operation describe \
   --topic paa.public. --resource-pattern-type prefixed
```

Once done with this, you need to restart the services:

```
sudo ncc services restart
```

To verify that a client can establish a secure connection, run the following command on an external client computer (not on the Control Center server). Below, `PUBLIC_HOSTNAME` is the Control Center hostname:

```
openssl s_client -debug -connect ${PUBLIC_HOSTNAME}:9093 -tls1_2 | grep "Secure Renegotiation IS supported"
```

In the command output you should see the server certificate as well as the following:

```
Secure Renegotiation IS supported
```

To ensure that internal services have been granted access to the Kafka server, please check the following logfiles:

- `/var/log/kafka/server.log`
- `/var/log/kafka/kafka-authorizer.log`

Validating External Client Connectivity

kafkacat

NOTE: These instructions are to be run on a *client computer* (not on the Control Center server).

NOTE: To display metrics information, ensure that at least one monitor is running in Control Center.

To verify and validate connectivity as an external client, it is possible to use the `kafkacat` utility which was installed in the section "Verifying That the Streaming API Works in Control Center" on page 4 .

Perform the following steps:

NOTE: Below, `CLIENT_USER` is the user previously specified in the file `/etc/kafka/server.properties` in Control Center: namely, `user_client` and the password set there.
The CA root certificate used to sign the server side SSL certificate must be present on the client.

- Create a file `client.properties` with the following content:

```
security.protocol=SASL_SSL
ssl.ca.location={PATH_TO_CA_CERT}
sasl.mechanisms=PLAIN
sasl.username={CLIENT_USER}
sasl.password={CLIENT_PASSWORD}
```

  where

  - `{PATH_TO_CA_CERT}` is the location of the CA root certificate used by the Kafka broker
  - `{CLIENT_USER}` and `{CLIENT_PASSWORD}` are the user credentials for the client.

- Run the following command to see the message consumed by `kafkacat`:

```
export KAFKA_FQDN=<Control Center hostname>
export METRICS_TOPIC=paa.public.accounts.<account short name>.metrics
```

```
kafkacat -b ${KAFKA_FQDN}:9093 -F client.properties -t ${METRICS_TOPIC} -C -e
```

where `{METRICS_TOPIC}` is the name of the Kafka topic with prefix `"paa.public."`.

NOTE: Older versions of `kafkacat` do not provide the `-F` option for reading the client settings from a file. If you are using such a version, you must provide the same settings from the command line as shown below.

```
kafkacat -b ${KAFKA_FQDN}:9093 \
-X security.protocol=SASL_SSL \
-X ssl.ca.location={PATH_TO_CA_CERT} \
-X sasl.mechanisms=PLAIN \
-X sasl.username={CLIENT_USER} \
-X sasl.password={CLIENT_PASSWORD} \
-t ${METRICS_TOPIC} -C -e
```

To debug the connectivity, you can use the `-d` option:

```
Debug consumer communications
kafkacat -d consumer -b ${KAFKA_FQDN}:9093 -F client.properties -t ${METRICS_TOPIC} -C -e

# Debug broker communications
kafkacat -d broker -b ${KAFKA_FQDN}:9093 -F client.properties -t ${METRICS_TOPIC} -C -e
```

Be sure to refer to the documentation for the Kafka client library in use, as the properties may differ from those in `client.properties`.

Message Format

The messages used for the metrics and metadata topics are serialized in the *Protocol buffers (protobuf)* format (see developers.google.com/protocol-buffers). The schemas for these messages adhere to the following format:

Metrics Protobuf Schema

```
syntax = "proto3";

import "google/protobuf/timestamp.proto";

package paa.streamingapi;
option go_package = ".;paa_streamingapi";

message Metrics {
  google.protobuf.Timestamp timestamp = 1;
  map<string, MetricValue> values = 2;
  int32 stream_id = 3;
}

/**
 * A metric value can be either an integer or a float.
 */
message MetricValue {

  oneof type {
    int64 int_val = 1;
    float float_val = 2;
  }
}
```

Metadata Protobuf Schema

```
syntax = "proto3";

package paa.streamingapi;
option go_package = ".;paa_streamingapi";

message Metadata {
  int32 stream_id = 1;
  string stream_name = 2;
  map <string, string>  tags = 13;
}
```

Client Examples

NOTE: These commands are intended to run on an external client, for example your laptop or similar, and not in Control Center.

NOTE: To have metrics information displayed, ensure that at least one monitor is running in Control Center.

The Control Center tarball includes the archive `paa-streaming-api-client-examples.tar.gz` (client-examples), which contains an example Python script showing how to use the Streaming API.

Installing and Configuring Client Examples

You find `client-examples` in the Paragon Active Assurance Control Center folder:

```
export CC_VERSION=4.4.0
cd ./paa-control-center_${CC_VERSION}
ls paa-streaming-api-client-examples*
```

To install `client-examples` on your external client computer, proceed as follows:

```
# Create directory for extracting the content of the client examples tarball
mkdir paa-streaming-api-client-examples

# Extract the content of the client examples tarball
tar xzf paa-streaming-api-client-examples.tar.gz -C paa-streaming-api-client-examples

# Go to the newly created directory
cd paa-streaming-api-client-examples
```

`client-examples` requires Docker to run. Downloads and installation instructions for Docker can be found at https://docs.docker.com/engine/install.

Using Client Examples

The `client-examples` tools can run in either basic or advanced mode to build examples of varying complexity. In both cases, it is also possible to run the examples with a configuration file containing additional properties for further customization of the client side.

Basic Mode

In basic mode, the metrics and their metadata are streamed separately. To this end, the client listens to each Kafka topic available for external access and simply prints the received messages to the console.

To start execution of the basic examples, run:

```
./build.sh run-basic --kafka-brokers localhost:9092 --account ACCOUNT_SHORTNAME
```

where `ACCOUNT_SHORTNAME` is the short name of the account you want to get the metrics from.

To terminate the execution of the example, press Ctrl + C. (There may be a slight delay before the execution stops because the client waits for a timeout event.)

Advanced Mode

NOTE: Metrics are displayed only for *HTTP monitors* running in Control Center.

Execution in advanced mode shows the correlation between metrics and metadata messages. This is possible thanks to the presence in each metrics message of a `stream id` field which refers to the corresponding metadata message.

To execute the advanced examples, run:

```
./build.sh run-advanced --kafka-brokers localhost:9092 --account ACCOUNT_SHORTNAME
```

where `ACCOUNT_SHORTNAME` is the short name of the account you want to get the metrics from.

To terminate the execution of the example, press Ctrl + C. (There may be a slight delay before the execution stops because the client waits for a timeout event.)

Additional Settings

It is possible to run the examples with additional configuration of the client using the `--config-file` option followed by a file name containing properties in the form `key=value`.

```
./build.sh run-advanced \
--kafka-brokers localhost:9092 \
--account ACCOUNT_SHORTNAME \
--config-file client_config.properties
```

NOTE: All files referenced in the command above must be located in the current directory and referred using only relative paths. This applies both to the `--config-file` argument and to all entries in the configuration file that describe file locations.

Validating External Client Authentication

To validate client authentication from outside the Control Center using `client-examples`, perform the following steps:

- From the Paragon Active Assurance Control Center folder, switch to the `paa-streaming-api-client-examples` folder:

  ```
  cd paa-streaming-api-client-examples
  ```

- Copy the CA root certificate `ca-cert` into the current directory.

- Create a `client.properties` file with the following content:

  ```
  security.protocol=SASL_SSL
  ssl.ca.location=ca-cert
  sasl.mechanism=PLAIN
  sasl.username={CLIENT_USER}
  sasl.password={CLIENT_PASSWORD}
  ```

  where `{CLIENT_USER}` and `{CLIENT_PASSWORD}` are the user credentials for the client.

- Run basic examples:

  ```
  export KAFKA_FQDN=<Control Center hostname>
  ./build.sh run-basic --kafka-brokers ${KAFKA_FQDN}:9093 \
  ```

```
--account ACCOUNT_SHORTNAME
--config-file client.properties
```

where `ACCOUNT_SHORTNAME` is the short name of the account you want to get the metrics from.

- Run advanced examples:

```
export KAFKA_FQDN=<Control Center hostname>
./build.sh run-advanced --kafka-brokers ${KAFKA_FQDN}:9093 \
--account ACCOUNT_SHORTNAME
--config-file client.properties
```

Appendix

In this appendix we describe how to create:

- a *keystore* file for storing the Kafka broker SSL certificate
- a *truststore* file for storing the Certificate Authority (CA) root certificate used to sign the Kafka broker certificate.

Creating a Kafka Broker Certificate

Creating a Certificate Using a Real Certificate Authority (Recommended)

It is recommended that you get a real SSL certificate from a trusted CA.

Once you have decided on a CA, copy their CA root certificate `ca-cert` file to your own path as shown below:

```
export CA_PATH=~/my-ca
mkdir ${CA_PATH}
cp ca-cert ${CA_PATH}
```

Create Your Own Certificate Authority

NOTE: Normally you should have your certificate signed by a real Certificate Authority; see the preceding subsection. What follows is just an example.

Here we create our own Certificate Authority (CA) root certificate file valid for 999 days (not recommended in production):

```
# Create a directory for storing the CA
export CA_PATH=~/my-ca
mkdir ${CA_PATH}

# Generate the CA certificate
openssl req -new -x509 -keyout ${CA_PATH}/ca-key -out ${CA_PATH}/ca-cert -days 999
```

Creating the Client Truststore

Now you can create a truststore file that contains the `ca-cert` generated above. This file will be needed by the Kafka client that will access the Streaming API:

```
keytool -keystore kafka.client.truststore.jks \
    -alias CARoot \
    -importcert -file ${CA_PATH}/ca-cert
```

Now that the CA certificate is in the truststore, the client will trust any certificate signed with it.

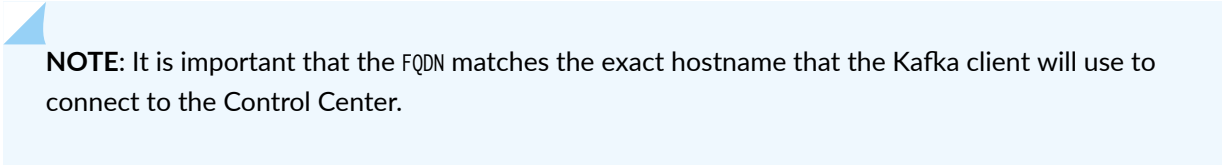
You should copy the file `kafka.client.truststore.jks` to a known location on your client computer and point to it in the settings.

Creating the Keystore for the Kafka Broker

To generate the Kafka broker SSL certificate and then the keystore `kafka.server.keystore.jks`, proceed as follows:

Generating the SSL Certificate

Below, 999 is the number of days of validity of the keystore, and `FQDN` is the fully qualified domain name of the client (public host name of the node).



NOTE: It is important that the `FQDN` matches the exact hostname that the Kafka client will use to connect to the Control Center.

```
sudo mkdir -p /var/ssl/private
sudo chown -R $USER: /var/ssl/private
```

```
cd /var/ssl/private

export FQDN=<Control Center hostname>

keytool -keystore kafka.server.keystore.jks \
    -alias server \
    -validity 999 \
    -genkey -keyalg RSA -ext SAN=dns:${FQDN}
```

Create a certificate signing request and store it in the file named `cert-server-request`:

```
keytool -keystore kafka.server.keystore.jks \
    -alias server \
    -certreq \
    -file cert-server-request
```

You should now send the file `cert-server-request` to your Certificate Authority (CA) if you are using a real one. They will then return the signed certificate. We will refer to this as `cert-server-signed` below.

Signing the SSL Certificate Using a Self-created CA Certificate

NOTE: Again, using your own CA is not recommended in a production system.

Sign the certificate using the CA by means of the file `cert-server-request`, which produces the signed certificate `cert-server-signed`. See below; `ca-password` is the password set when creating the CA certificate.

```
cd /var/ssl/private
openssl x509 -req \
    -CA ${CA_PATH}/ca-cert \
    -CAkey ${CA_PATH}/ca-key \
    -in cert-server-request \
    -out cert-server-signed \
    -days 999 -CAcreateserial \
    -passin pass:{ca-password}
```

Importing the Signed Certificate into the Keystore

Import the `ca-cert` root certificate into the keystore:

```
keytool -keystore kafka.server.keystore.jks \
          -alias ca-cert \
          -import \
          -file ${CA_PATH}/ca-cert
```

Import the signed certificate referred to as `cert-server-signed`:

```
keytool -keystore kafka.server.keystore.jks \
    -alias server \
    -import \
    -file cert-server-signed
```

The file `kafka.server.keystore.jks` should be copied to a known location on the Control Center server, and then referred to in `/etc/kafka/server.properties`.

Using the Streaming API

IN THIS SECTION

General

The streaming API fetches both test and monitor data. It is not possible to single out one of these categories.

The streaming API does not fetch data from script-based tests (those represented by a rectangle instead of a jigsaw piece in the Control Center GUI), such as Ethernet service activation tests and transparency tests.

Kafka Topic Names

The Kafka topic names for the streaming API are as follows, where `%s` is the short name of the Control Center account (indicated when creating the account):

```
const (
        exporterName     = "kafka"
        metadataTopicTpl = "paa.public.accounts.%s.metadata"
        metricsTopicTpl  = "paa.public.accounts.%s.metrics"
        )
```

Examples of Using the Streaming API

The examples that follow are found in the tarball `paa-streaming-api-client-examples.tar.gz` contained within the Control Center tarball.

First, there is a basic example demonstrating how the metrics and their metadata are streamed separately and simply print the received messages to the console. You can run it as follows:

```
sudo ./build.sh run-basic --kafka-brokers localhost:9092 --account ACCOUNT_SHORTNAME
```

There is also a more advanced example where metrics and metadata messages are correlated. Use this command to run it:

```
sudo ./build.sh run-advanced --kafka-brokers localhost:9092 --account ACCOUNT_SHORTNAME
```

You need to use `sudo` to run Docker commands such as the ones above. Optionally, you can follow the Linux post-installation steps to be able to run Docker commands without `sudo`. For details, go to docs.docker.com/engine/install/linux-postinstall.