EVERTRUST

Stream

Version 2.2, 2026-11-10

Table of Contents

1. Installation . . . 1
   1.1. Introduction . . . 1
   1.2. Installing on Linux . . . 1
   1.3. Installing on Kubernetes . . . 53
   1.4. Tinkey . . . 82
   1.5. Secret Rotator . . . 96
   1.6. Monitoring . . . 98
   1.7. Troubleshooting . . . 101
   1.8. Advanced configuration . . . 105
   1.9. Upgrade . . . 120
   1.10. Uninstallation . . . 124
2. Admin guide . . . 127
   2.1. Introduction . . . 127
   2.2. Managing Certification Authorities . . . 127
   2.3. Managing Certificate Revocation . . . 132
   2.4. Managing Certificate Templates & EKUs . . . 140
   2.5. Managing Certificate Lifecycle . . . 142
   2.6. OpenSSH . . . 144
   2.7. Managing Keystores & Keys . . . 147
   2.8. Managing Notifications . . . 152
   2.9. Managing Security . . . 156
   2.10. Managing Stream instance . . . 164
   2.11. Timestamping . . . 173
   2.12. Backup and Restore . . . 174
   2.13. Dictionaries . . . 177
   2.14. Computation rule . . . 180
   2.15. Template Strings . . . 188

Chapter 1. Installation

1.1. Introduction

Description

Stream version 2.2 is EVERTRUST Certification Authority. This document is an installation procedure detailing how to install and bootstrap a Stream instance on your infrastructure for the version 2.2. It does not describe how to configure and operate the instance. Please refer to the administration guide for administration related tasks.

Prerequisites

Choose an installation method

We offer three installation modes:

- Enterprise Linux 8.x/9.x x64
- Debian based Linux x64
- A cloud-native installation using Kubernetes

Depending on your needs, you'll have to choose the solution that fits your use cases the best. Reach out to our support team to get suggestions on how to deploy on your infrastructure.

Gathering your credentials

Both methods require that you download the binaries of the Stream software from our software repository. The access to this repository is protected by username and password, which you should have got from our tech team. If you don't, you won't be able to continue with the installation. Email us to get your credentials, and come back to this step.

1.2. Installing on Linux

1.2.1. Pre-requisites

This section describes the system and software pre-requisites to install Stream. Please select your OS type in the available tabs.

System pre-requisites

The following elements are considered as system pre-requisites:

RHEL

- A server running a `en-US` minimal install EL [8.x-9.x] x64 (CentOS / RHEL) with the

network configured.

- Base EL [8.x-9.x] x64 repositories activated;
- An access with administrative privileges (root) to the server mentioned above as most commands are system-related and require super user privilege;
- Installed and configured Java 17 or 21;

Debian

- A server running Debian 11/12 or Ubuntu 22/24 x64 with the network configured and **AppArmor** as well as **FIPS mode** disabled;
- Standard Debian/Ubuntu repositories activated;
- An access with administrative privileges (root) to the server mentioned above;
- Installed and configured Java 17 or 21;

Software pre-requisites

All the following packages can be necessary to deploy Stream. Most are available on public repositories but some require specific configurations.

Package name	Mandatory	Online Instructions	Offline instructions	Additional information
stream-2.2.x package	☑	Online steps	Offline steps	
mongodb-mongosh, mongod-org-server-mongodb-org-tools, mongodb-database-tools `, mongodb-org-database-tools-extra	☑	Online steps	Offline steps	
nginx		Online steps	N/A	Recommended reverse proxy
stream-hardening-1.x86_64.rpm (RHEL Only)		Online steps	Offline steps	Configuration hardening rpm

1.2.2. Installation

Installing MongoDB



Stream requires at least MongoDB version 4.4.2. For support reasons, EVERTRUST recommends to use the latest available version, which is MongoDB 8 at the time of writing.

Stream relies on MongoDB to store its data, whether it be configuration elements or certificate data. The necessary packages are `mongodb-org-server`, `mongodb-mongosh`, `mongodb-org-tools`, `mongodb-database-tools` and `mongodb-org-database-tools-extra`. To install and configure MongoDB, follow these steps using an account with administrative privileges:

Installation with Internet Access

These steps are for when the server has internet access

RHEL

1. Follow step 1 of the official MongoDB installation tutorial.

2. Run the following command to install the RPMs:

```
# yum install -y mongodb-org-server mongodb-mongosh mongodb-org-tools mongodb-
org-database-tools-extra mongodb-database-tools
```

Debian

1. Follow step 1 through 3 of the official MongoDB installation tutorial.

2. Run the following command to install the DEBs:

```
# apt install -y mongodb-org-server mongodb-mongosh mongodb-org-tools mongodb-
org-database-tools-extra mongodb-database-tools
```

Installation without Internet Access

RHEL

1. Download the .rpm files directly from the MongoDB repository. Downloads are organized by Red Hat / CentOS version (e.g. 8 - do not select the Server folders), then MongoDB release version (e.g. 8.0), then architecture (e.g. x86_64). Upload the files to the server.

2. Run the following command to install the RPMs:

```
# yum localinstall mongodb-org-server-x.y.z.arch.rpm mongodb-mongosh-
x.y.z.arch.rpm mongodb-org-tools-x.y.z.arch.rpm mongodb-org-database-tools-
extra-x.y.z.arch.rpm mongodb-database-tools-x.y.z.arch.rpm
```

Debian

1. Download the .deb packages directly from the MongoDB APT repository. Downloads are organized by Debian version (e.g. bookworm - do not select the Server folders), then MongoDB release version (e.g. 8.0), then architecture (e.g. x86_64). Upload the files to the server.

2. Run the following command to install the DEBs:

```
# apt install -y mongodb-org-server_8.0.x_amd64.deb mongodb-
mongosh_2.x_amd64.deb mongodb-org-tools_8.0.x_amd64.deb mongodb-org-database-
```

```
tools-extra_8.0.x_amd64.deb mongodb-database-tools_x_amd64.deb
```

Common installation steps

3. Enable the service at startup with the following command:

```
# systemctl enable mongod
```

4. Start the `mongod` service with the following command:

```
# systemctl start mongod
```

5. Start the `mongosh` executable using the following command to check that the database is up and running:

```
# mongosh
```

For now, since we did not set up access control, everyone using `localhost` as DB URI can connect as administrator, which is something that needs to be prevented before setting-up Stream.



The following section is not mandatory to get Stream up and running, but is highly recommended for security purposes.

6. In the mongo shell that was just opened, run the following commands:

```
> use admin;
> db.createUser(
  {
    user: "stream_db_admin",
    pwd:  "AComplexPassword",
    roles: [ { role: "dbOwner", db: "stream" } ]
  }
)
```

This way, the created `stream_db_admin` user has owner permissions on the database named `stream`. You can change the `stream_db_admin` value to what you want to use as database username, the password to be what you want to use as a database password to match your password policies and the database name (the value to the `db` key) to what you want to use as the stream database. For the password, you can also `passwordPrompt()` (without quotes) as the password value, which will prompt you for a password upon pressing Enter. Be careful though as this is a password prompt without confirmation.



If you plan on using special characters in the password, be careful as the MongoDB

engine has trouble with some of them. For more information on this topic, please refer to the MongoDB documentation.

7. Edit the `/etc/mongod.conf` file and add the following section at the end:

```
security:
  authorization: enabled

setParameter:
  enableLocalhostAuthBypass: false
```

These options will prevent anonymous login to the MongoDB instance and will disable the localhost bypass.

8. Restart the MongoDB daemon to make the changes effective:

```
# systemctl restart mongod
```

9. When setting up Stream, use this connection string as the MongoDB URI :

```
mongodb://stream_db_admin:AComplexPassword@127.0.0.1:27017/stream?authSource=admin
```

If you used another username for the MongoDB user, replace the `stream_db_admin` part with the username that you used. Replace the `AComplexPassword` in the URI by the password that you chose when creating the account.
Replace `/stream` in the URI by `/databaseName` if you chose to use another name for your Stream database when creating the user.

Installing NGINX



In order to install Stream, the server must have access to a repository (mirror, ISO file, ...) of the linux distribution you are using in order to be able to install the dependencies of the software.

1. Connect to the server with an account with administrative privileges;
2. Install the NGINX web server using the following command:

RHEL

```
# yum install nginx
```

Debian

```
# apt install nginx
```

3. Enable NGINX to start at boot using the following command:

```
# systemctl enable nginx
```

4. Stop the NGINX service with the following command:

```
# systemctl stop nginx
```

Install Stream

RHEL



In order to install Stream, the server must have access to a repository (mirror, ISO file, ...) of the linux distribution you are using in order to be able to install the dependencies of the software. Stream package has the following dependencies:

- `dialog`
- `tzdata-java`
- `zip`
- `unzip`
- `tar`

Please note that these packages may have their own dependencies.

Debian



In order to install Stream, the server must have access to a repository (mirror, ISO file, ...) of the linux distribution you are using in order to be able to install the dependencies of the software. Stream package has the following dependencies:

- `dialog`
- `zip`
- `unzip`
- `tar`

Please note that these packages may have their own dependencies.

Installation from the EverTrust repository

RHEL

Create a `/etc/yum.repos.d/stream.repo` file containing the EverTrust repository info:

```
[stream]
enabled=1
name=Stream Repository
baseurl=https://repo.evertrust.io/repository/stream-rpm/
gpgcheck=1
gpgkey=https://evertrust.io/.well-known/rpm/gpg.pub
username=<username>
password=<password>
```

Replace `<username>` and `<password>` with the credentials you were provided.

You can then run the following to install the latest Stream version:

```
# yum install stream
```

To prevent unattended upgrades when running yum update, you should pin the Stream version by adding

```
exclude=stream
```

at the end of the `/etc/yum.repos.d/stream.repo` file after installing Stream.

Debian

If you haven't already, to add the EVERTRUST repository to your APT repositories, run the following commands:

1. Install the required tools (`gpg`)

```
# sudo apt install gnupg
```

2. Download and install the EVERTRUST GPG key

```
# curl https://evertrust.io/.well-known/apt/gpg.pub | sudo gpg -o
/usr/share/keyrings/evertrust.gpg --dearmor
```

3. Add the repository

```
# echo "deb [ arch=all signed-by=/usr/share/keyrings/evertrust.gpg ]
https://repo.evertrust.io/repository/apt all main" | sudo tee
/etc/apt/sources.list.d/evertrust.list
```

Once the repository has been added, authentication to it must be provided. To do so, edit the `/etc/apt/auth.conf` file and add the following lines:

```
machine repo.evertrust.io
login <your EVERTRUST login>
password <your EVERTRUST password>
```

Once the repository has been added, run the following command to update the APT repository list.

```
# sudo apt update
```

You can then run the following command to install the latest Stream version:

```
# sudo apt install stream
```

To prevent unattended upgrades when running `apt upgrade`, you should pin the Stream version by creating a `/etc/apt/preferences.d/stream` file:

```
Package: stream
Pin: version <installed-version>
Pin-Priority: 1001
```

Installing from the package file

RHEL

Download the latest RPM for Stream on the Official EVERTRUST repository.

Upload the file '***stream-<latest>.noarch.rpm***' to the server;

Access the server with an account with administrative privileges;

Install the Stream package with the following command:

```
# yum localinstall /root/stream-<latest>.noarch.rpm
```

If you wish to verify the signature of the RPM package, the EVERTRUST key can be added to your trusted keys using the following command:

```
# rpm --import https://evertrust.io/.well-known/rpm/gpg.pub
```

The signature can then be verified using the following command:

```
# rpm -K /root/stream-<latest>.noarch.rpm
```

Debian

Download the latest DEB for Stream on the Official EVERTRUST repository.

Upload the file '***stream-<latest>_all.deb***' to the server;

Access the server with an account with administrative privileges;

Install the Stream package with the following command:

```
# apt install /root/stream-<latest>_all.deb
```

Enabling the service

After installing, services must be reloaded to take the change into account

```
# systemctl daemon-reload
```

Enable Stream to start at boot using the following command

```
# systemctl enable stream
```

1.2.3. Configuration

Initial Configuration

Introduction

This section assumes that Stream is running in a confined environment: nobody but the person performing the configuration operation and the key ceremony stakeholders should have access to Stream yet, and they should do so under the supervision of a security officer.

RHEL

If `selinux` is enabled, it should be disabled during the initial configuration and bootstrapping operations. It will be re-enabled following the security guidelines.

```
# setenforce Permissive
```

To ensure that it is permissive, run the following command

```
# getenforce
```

This should return `Permissive`

Debian

Stream installation requires **AppArmor** as well as **FIPS mode** to be disabled.

Configuring the firewall



In order for Stream to work properly, the following ports are used:

- Exposed: 443 for HTTPS access to the product (through the web interface or through the API);
- Exposed: 80 for HTTP access to the product only to retrieve CRLs (the only allowed endpoint must be `/crls/*`, this is the case for the default NGINX configuration);
- Internal: 25520 and 7626 for high-availability configurations through the Pekko framework.
- Internal: 9000 for the Stream API.

RHEL

Connect to the server with an account with administrative privileges;

Open port TCP/443 on the local firewall with the following command:

```
# firewall-cmd --permanent --add-service=https
```

Stream also needs HTTP traffic allowed since it is required to set up the CRLDPs :

```
# firewall-cmd --permanent --add-service=http
```

To make the change effective, you need to restart the firewall service:

```
# systemctl restart firewalld
```

Enable the service at startup with the following command:

```
# systemctl enable firewalld
```

Debian

Make sure to allow the HTTP and HTTPS traffic if you have any firewall set up.

Generating a Tink keyset

To protect its secrets, Stream relies on Tink. A Tink keyset can be issued as:

- A plaintext keyset (stored as a file, protected by the filesystem rights and SELinux);
- A GCP keyset (protected by a master key in a GCP KMS);
- An AWS keyset (protected by a master key in an AWS KMS).
- A PKCS#11 keyset (protected by a master key in an HSM).



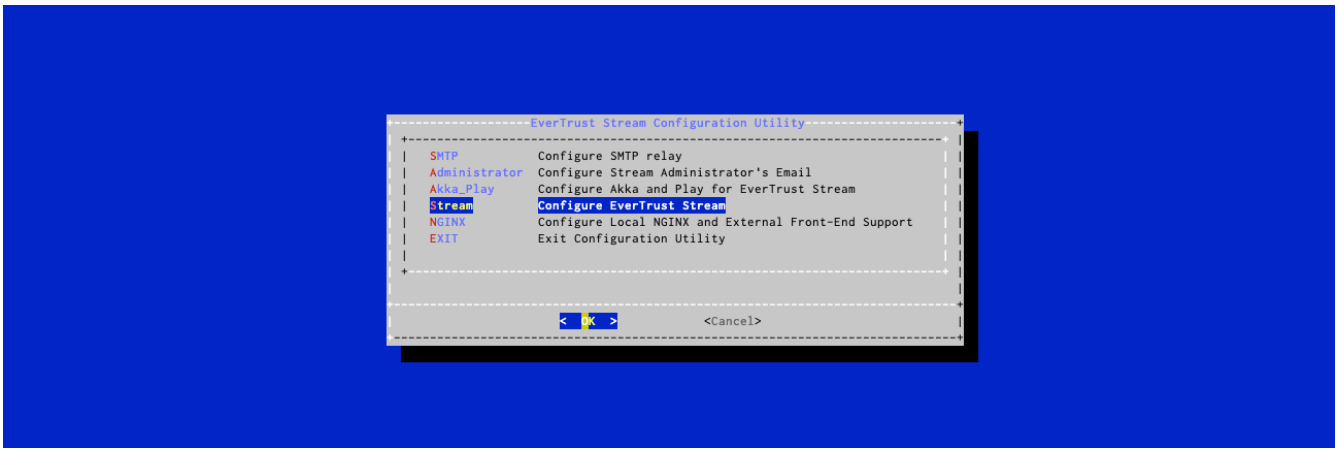
In order to generate a keyset, the Tinkey tool must be installed.

Connect to the server with an account with administrative privileges;

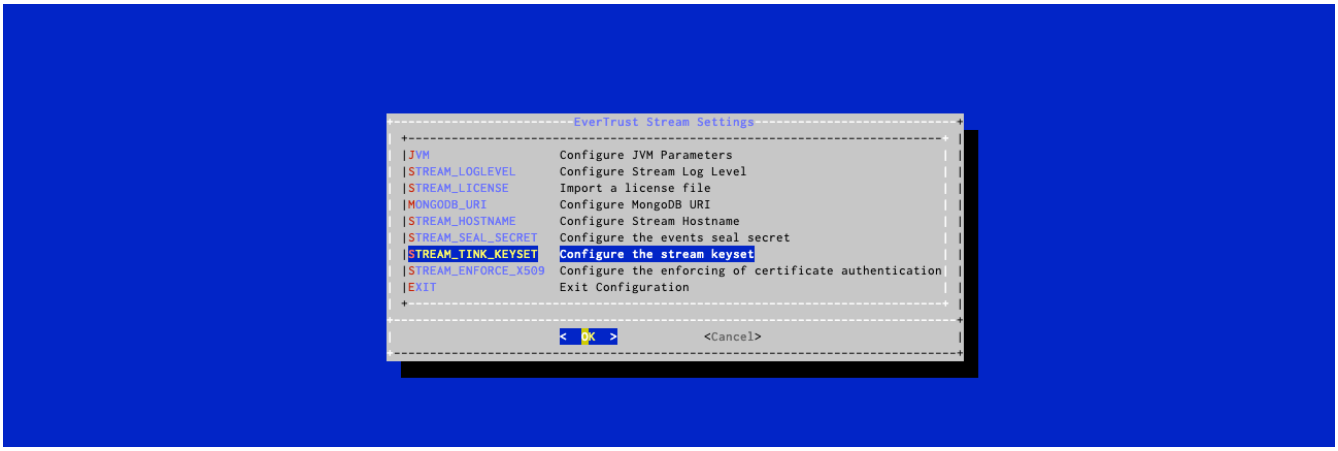
Start the Stream configuration utility by running:

```
# /opt/stream/sbin/stream-config
```

In the main menu, select '**Stream**':

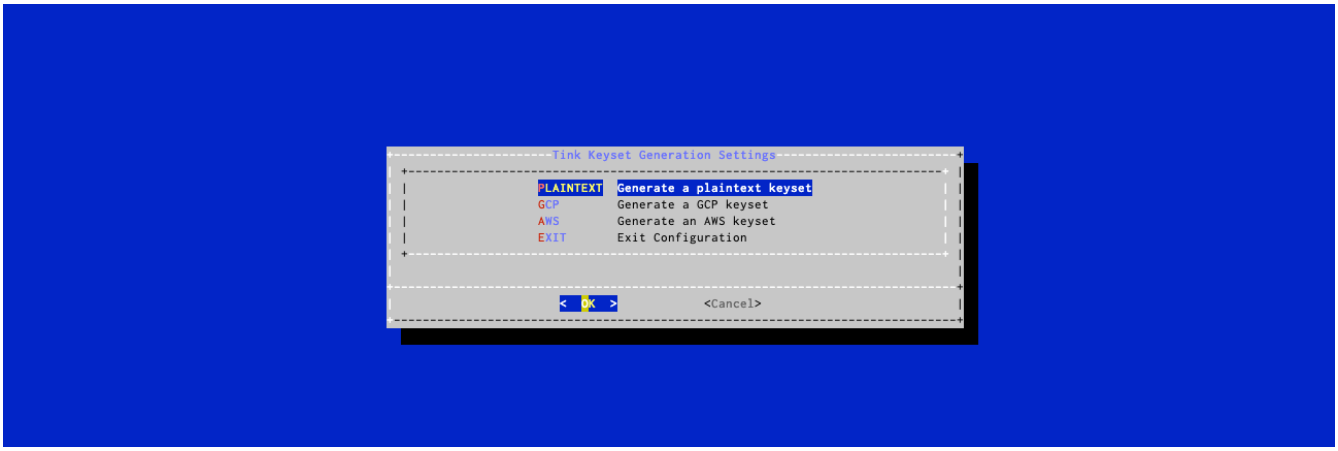


In the Stream menu, select '**STREAM_TINK_KEYSET**':



Generating a plaintext keyset

In the Tink Keyset Generation menu, select '**PLAINTEXT**':



The keyset will be generated automatically. For the changes to take effect, you must restart the Stream service by running:

```
# systemctl restart stream
```

Generating a GCP protected keyset

In the Tink Keyset Generation menu, select '**GCP**':



The URL of the GCP master key must be typed in the menu.

After pressing **OK**, the keyset will be generated automatically. For the changes to take effect, you must restart the Stream service by running:

```
# systemctl restart stream
```

Generating an AWS protected keyset

In the Tink Keyset Generation menu, select '**AWS**':



The URL of the AWS master key must be typed in the menu.

After pressing **OK**, the keyset will be generated automatically. For the changes to take effect, you must restart the Stream service by running:

```
# systemctl restart stream
```

Generating a PKCS#11 protected keyset

In the Tink Keyset Generation menu, select '**PKCS11**':

The URL of the PKCS#11 master key must be typed in the menu.

The expected format is:

```
pkcs11://object=<object name>;type=<object type>;slot-id=<slot id>?module-
path=<library path>&pin-value=<pin>;
```

Example:

```
pkcs11://object=kek;type=secret-key;slot-id=-1?module-
path=/usr/lib/softhsm/libsofthsm2.so&pin-value=1234";
```

After pressing **OK**, the keyset will be generated automatically. For the changes to take effect, you must restart the Stream service by running:

```
# systemctl restart stream
```

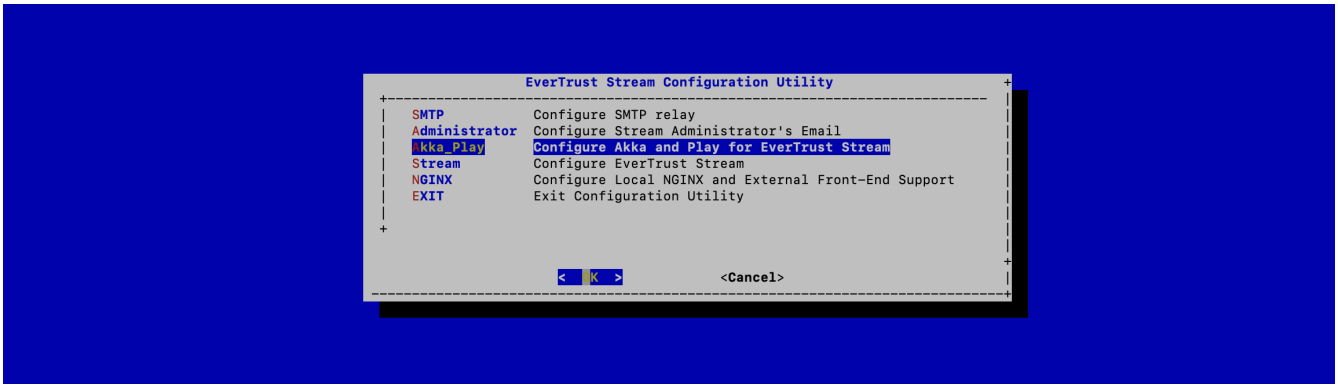
Generating a Play secret

Connect to the server with an account with administrative privileges;

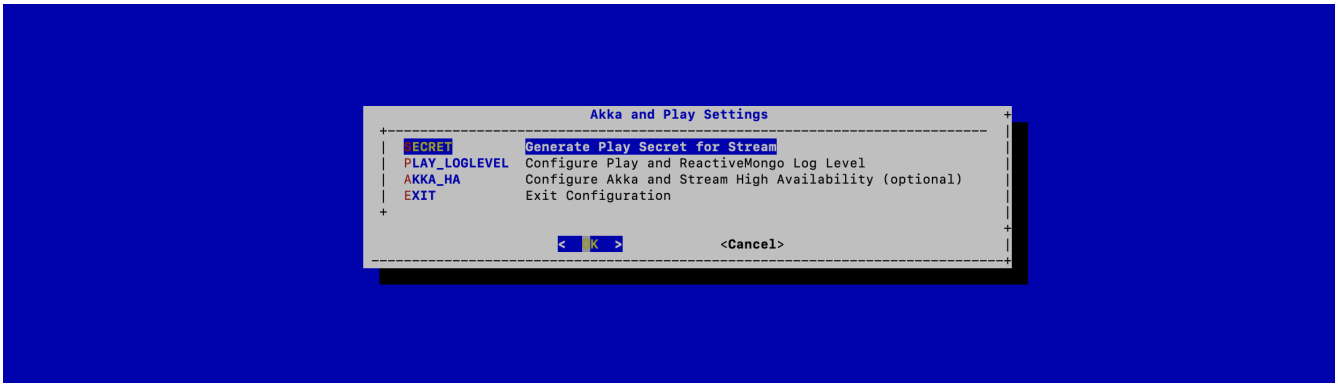
Start the Stream configuration utility by running:

```
# /opt/stream/sbin/stream-config
```

In the main menu, select '**Pekko_Play**':



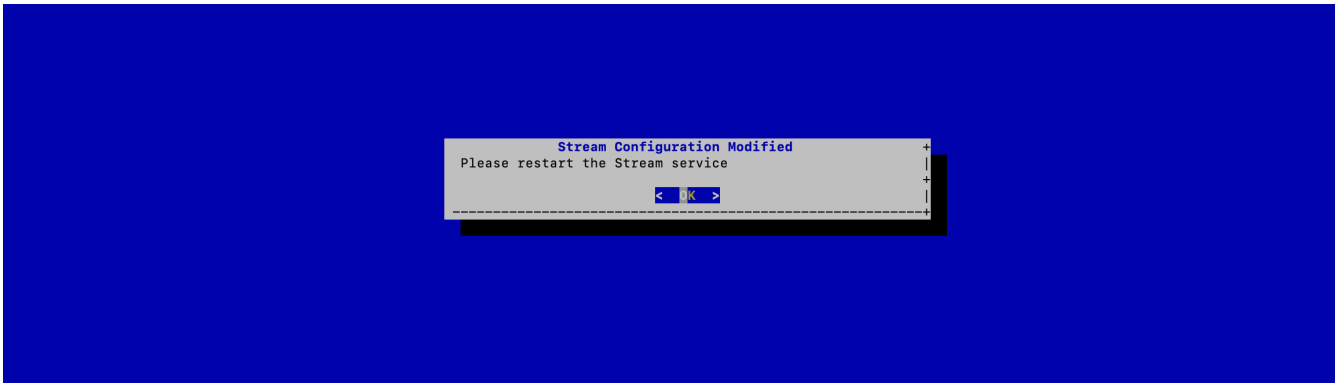
In the Pekko_Play menu, select '**SECRET**':



Validate the new Stream Application Secret:



The Stream configuration is updated:



For the changes to take effect, you must restart the Stream service by running:

```
# systemctl restart stream
```

JVM Configuration

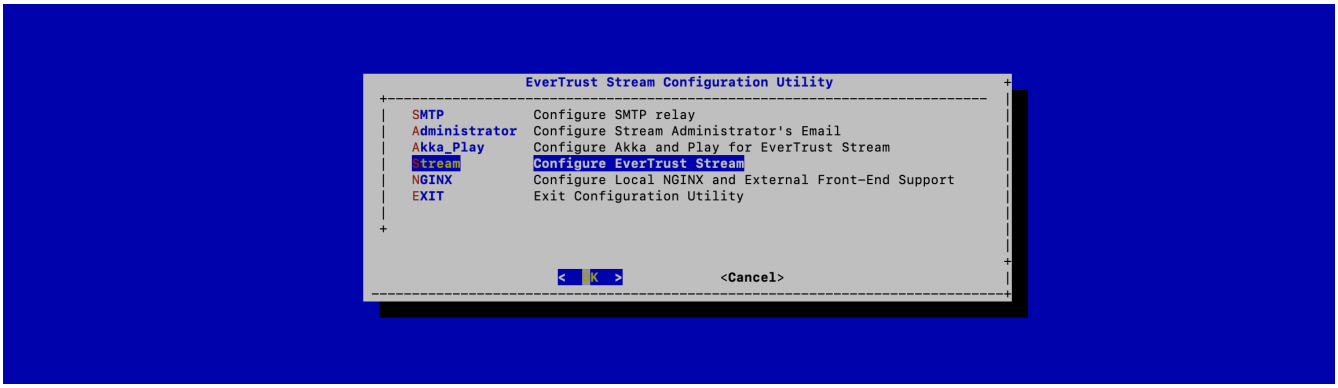
Stream allows you to configure the ***Xms*** (minimum memory allocation pool) and ***Xmx*** (maximum memory allocation pool) parameters of the JVM running Stream using the configuration tool.

Connect to the server with an account with administrative privileges;

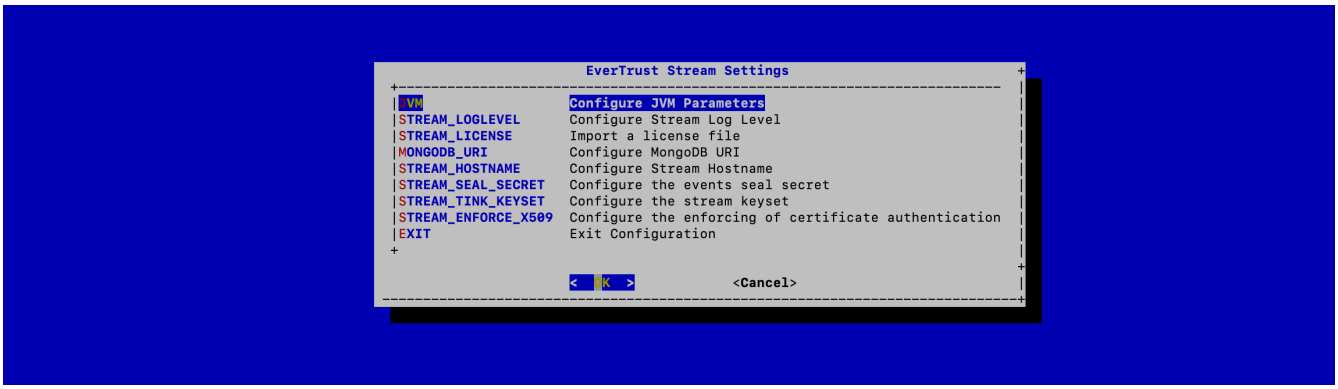
Start the Stream configuration utility by running:

```
# /opt/stream/sbin/stream-config
```

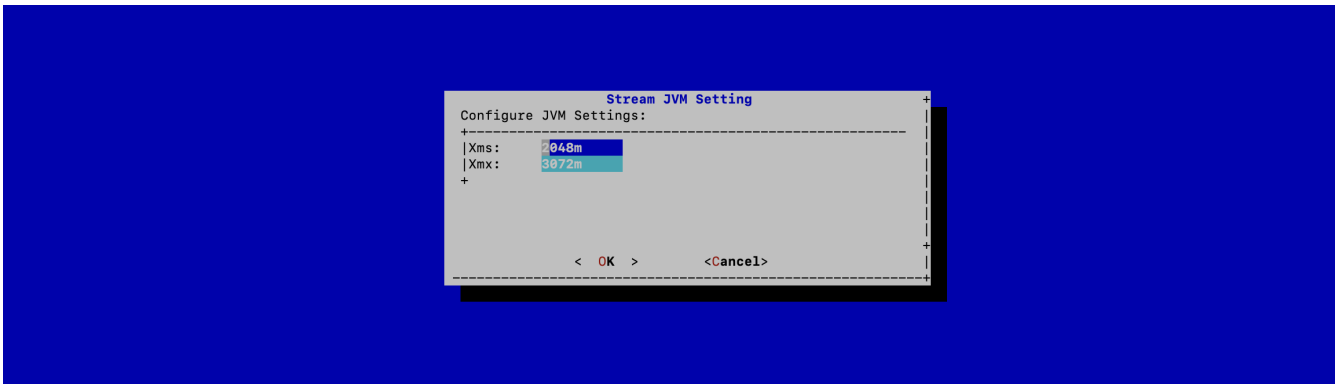
In the configuration menu, select **Stream**:



In the Stream configuration menu, Select **JVM**:



Specify the 2048 for ***xms*** and 3072 for ***xmx*** parameters and select '**OK**':



The new JVM parameters are configured.

For the changes to take effect, you must restart the Stream service by running:

```
# systemctl restart stream
```

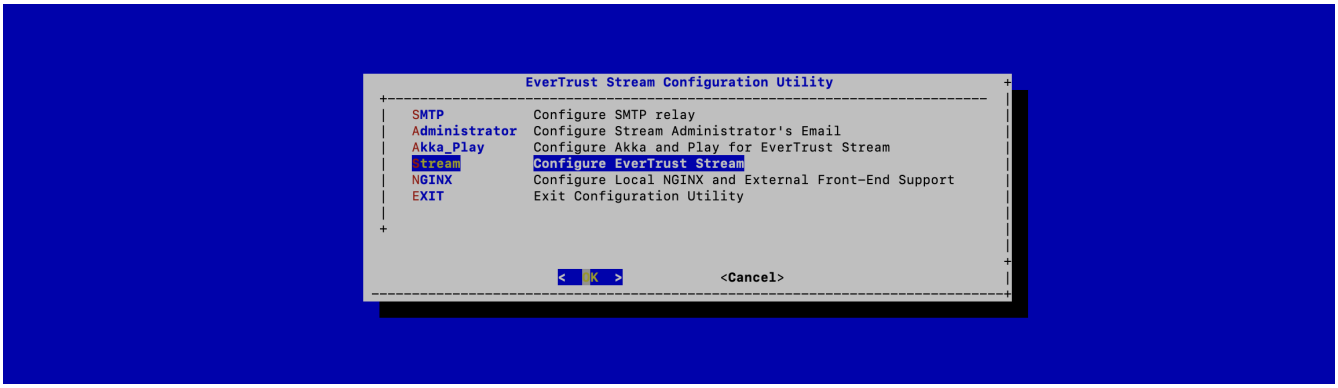
MongoDB URI Configuration

Connect to the server with an account with administrative privileges;

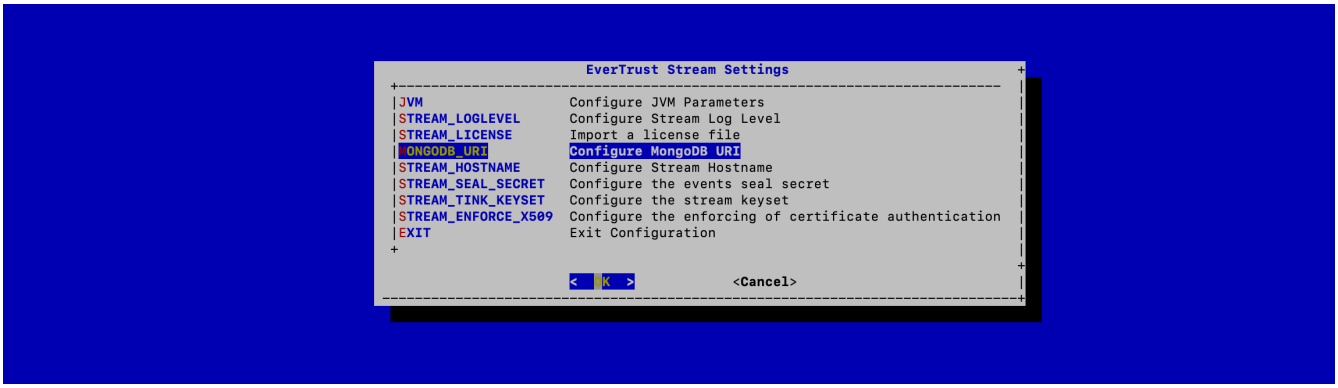
Start the Stream configuration utility by running:

```
# /opt/stream/sbin/stream-config
```

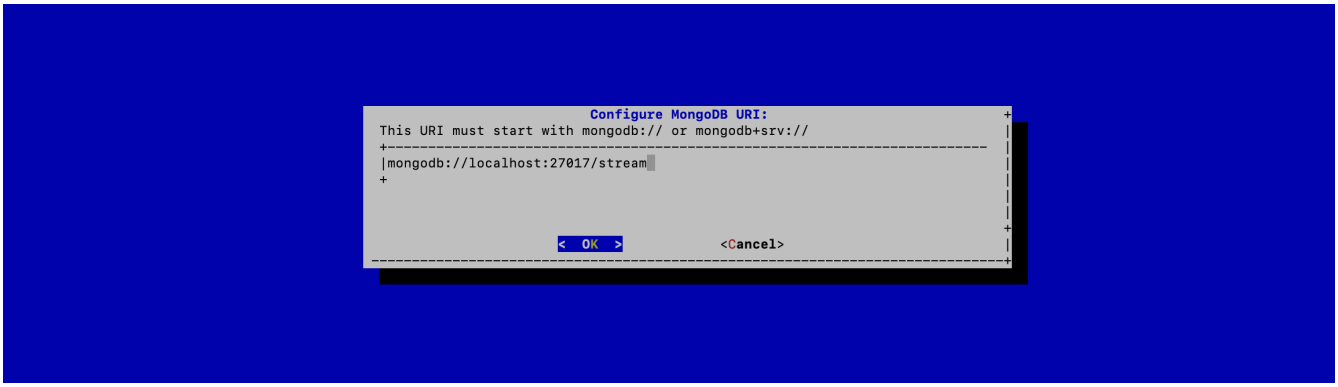
In the main menu, select **Stream**:



In the Stream configuration menu, Select **MONGODB_URI**:



Specify the MongoDB URI to target your MongoDB instance:





Stream is installed to target a local MongoDB instance by default.

If you use an external MongoDB (such as MongoDB Atlas Database or dedicated On-premises database) instance:

- Create a user with "read/write" permissions on your MongoDB instance;
- Create a replicaSet if using a MongoDB cluster;
- Specify a MongoDB URI that does match your context.

External MongoDB database URI syntax

```
mongodb://<user>:<password>@<hostname>:<port>/stream
```

External MongoDB cluster of databases URI syntax

```
mongodb://<user>:<password>@<hostname1>:<port1>,<hostname-
2>:<port2>/stream?replicatSet=<replicaset>&authSource=admin
```

The MongoURI is configured.

For the changes to take effect, you must restart the Stream service by running:

```
# systemctl restart stream
```

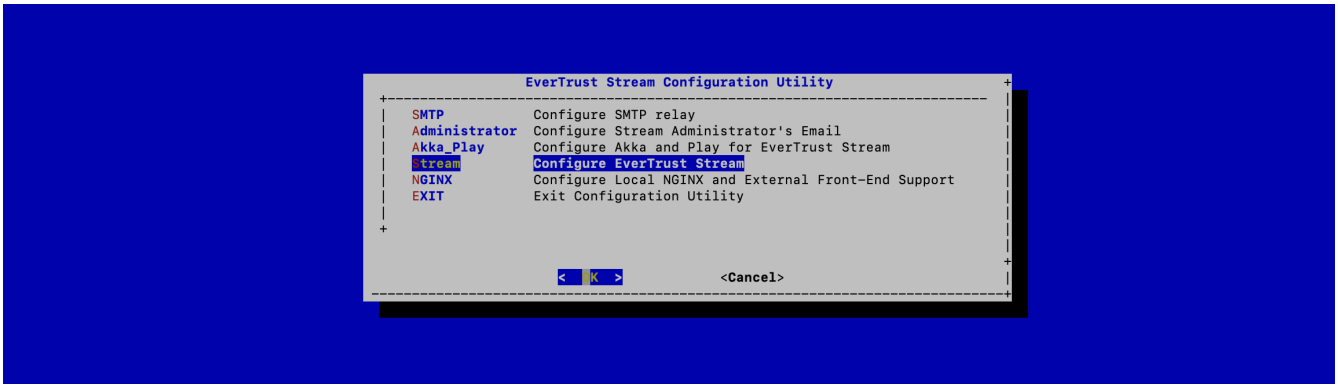
Stream Hostname Configuration

Connect to the server with an account with administrative privileges;

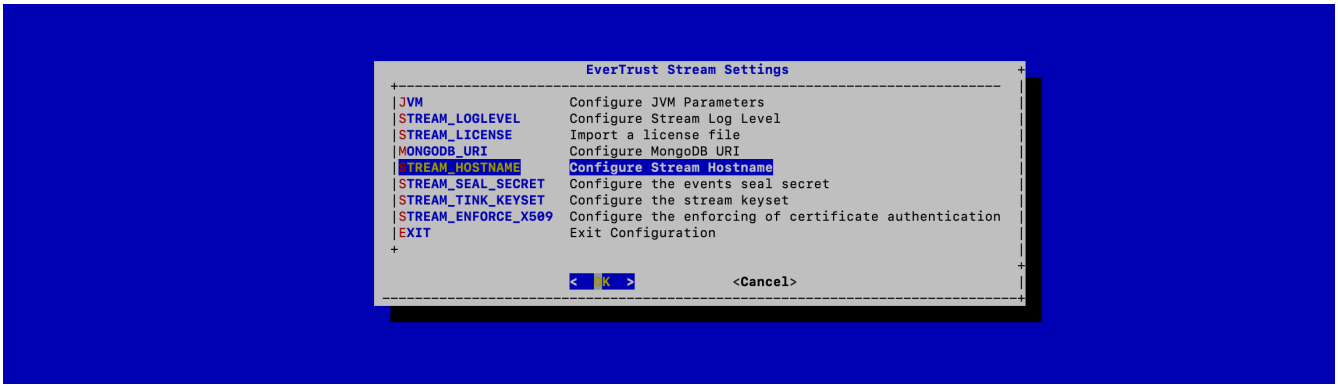
Start the Stream configuration utility by running:

```
# /opt/stream/sbin/stream-config
```

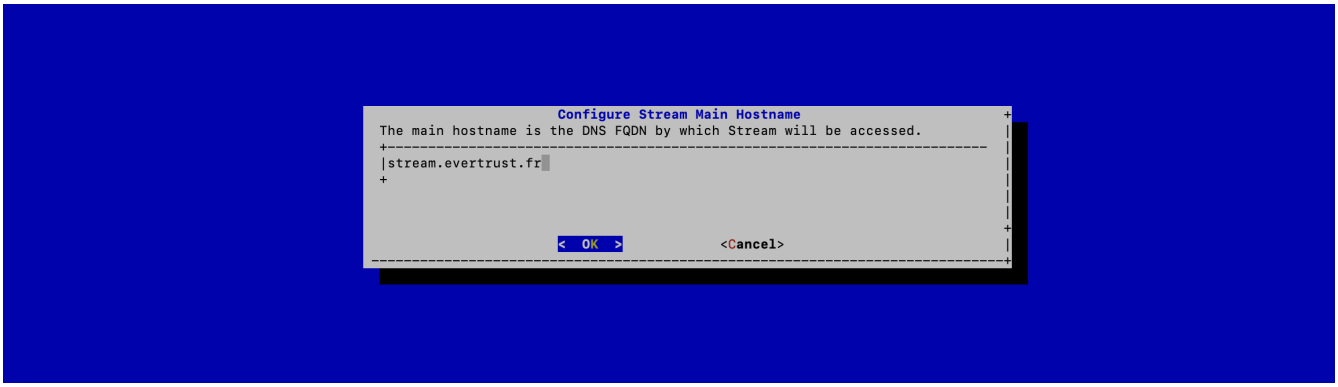
In the main menu, select **Stream**:



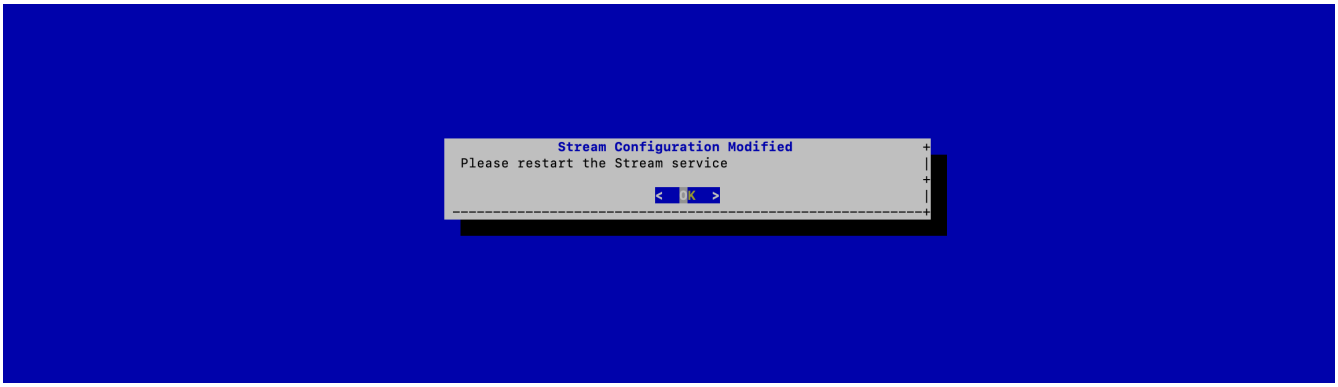
In the Stream configuration menu, Select **STREAM_HOSTNAME**:



Specify the DNS FQDN by which Stream will be accessed:



The Stream Hostname is configured:



For the changes to take effect, you must restart the Stream service by running:

```
# systemctl restart stream
```

Generating an event seal secret

Stream will generate functional events when using the software.

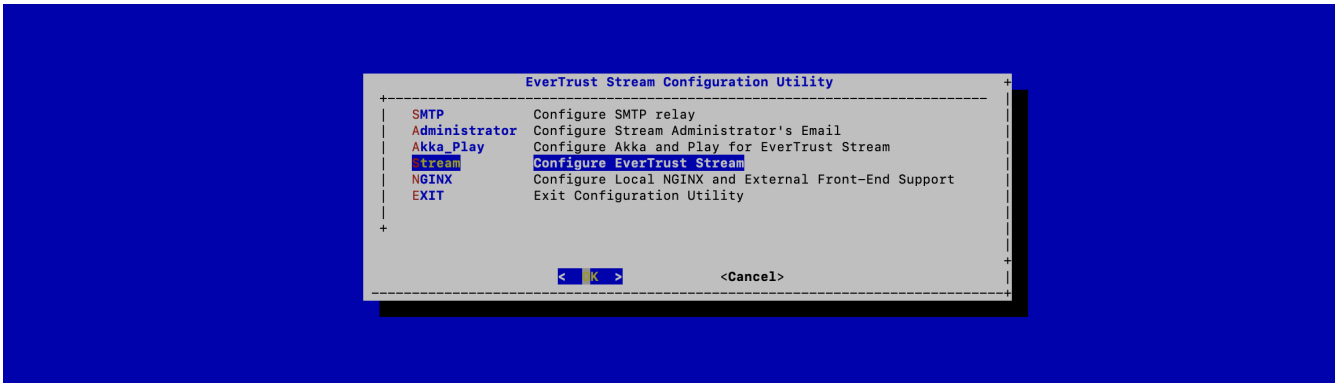
These events are typically signed and chained to ensure their integrity. Therefore, you must specify a sealing secret for this feature to work properly.

Connect to the server with an account with administrative privileges;

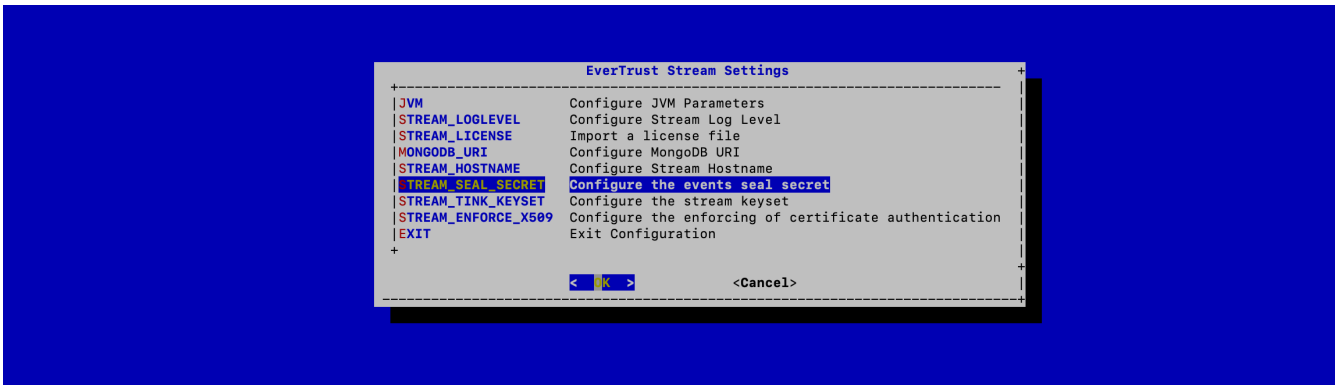
Start the Stream configuration utility by running:

```
# /opt/stream/sbin/stream-config
```

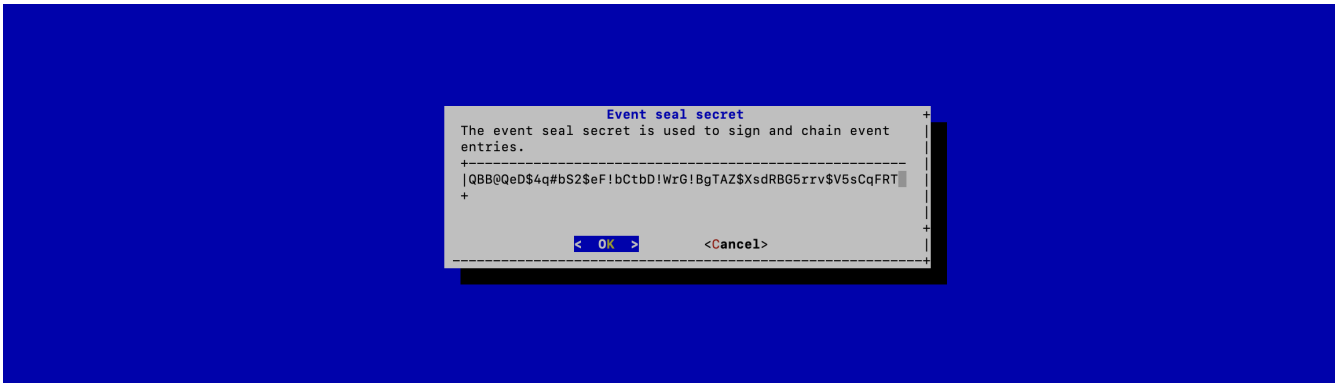
In the main menu, select '**Stream**':



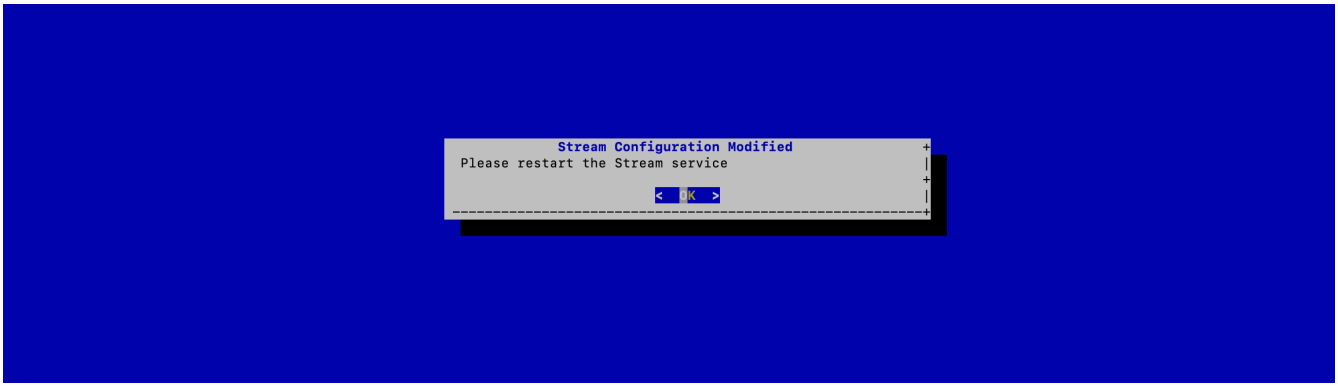
In the Stream menu, select '**STREAM_SEAL_SECRET**':



Validate the new event seal secret:



The even seal secret is now configured:



For the changes to take effect, you must restart the Stream service by running:

```
# systemctl restart stream
```

Installing the Stream license



You should have been provided with a `stream.lic` file. This file is a license file and indicates an end of support date.

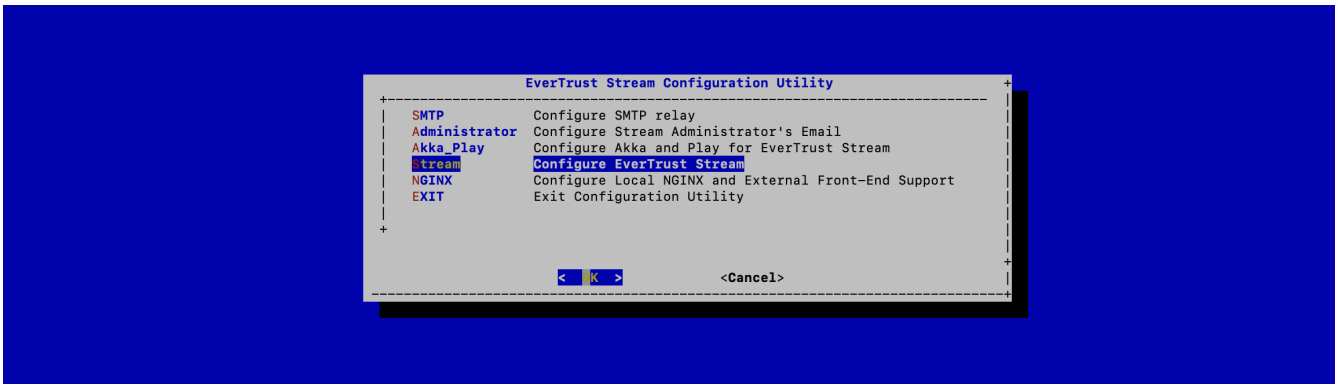
Upload the `stream.lic` file (using SCP or other means) under `/tmp/stream.lic`;

Connect to the server with an account with administrative privileges;

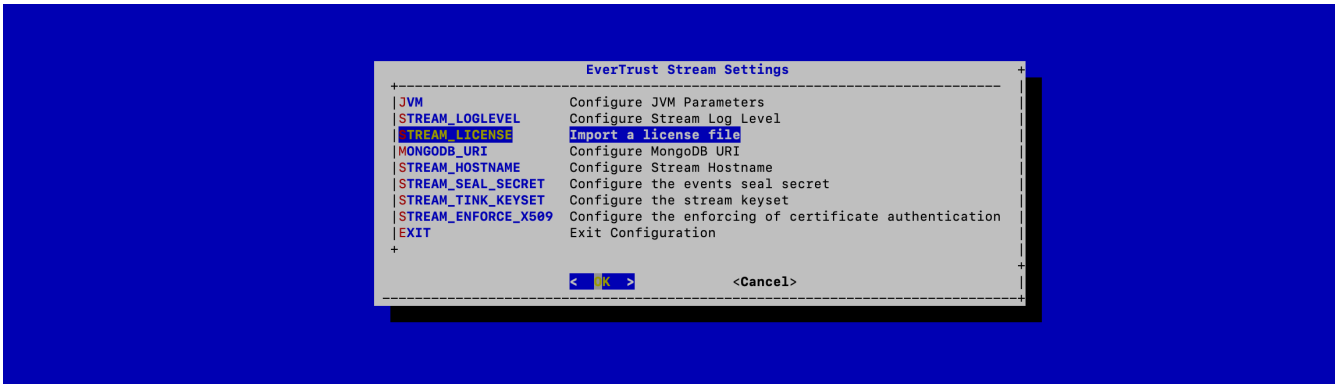
Start the Stream configuration utility by running:

```
# /opt/stream/sbin/stream-config
```

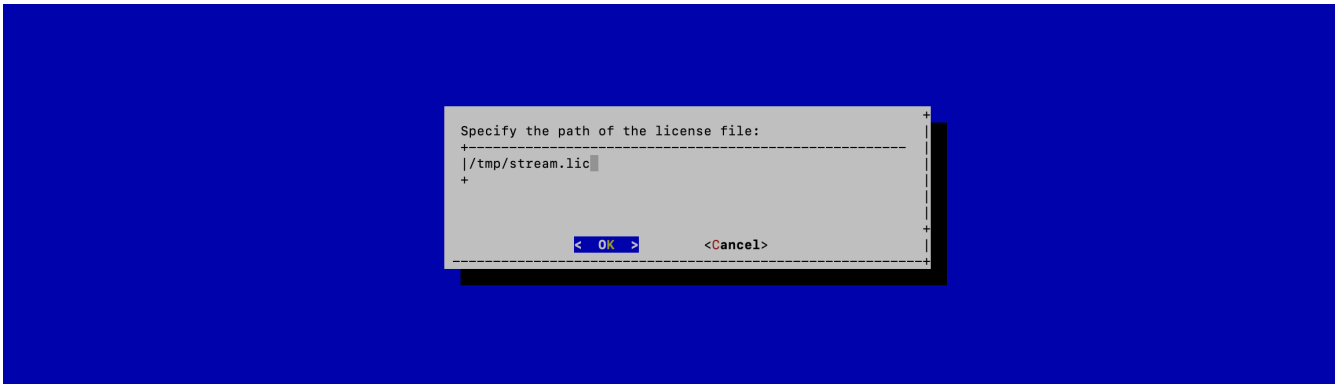
In the main menu, select **Stream**:



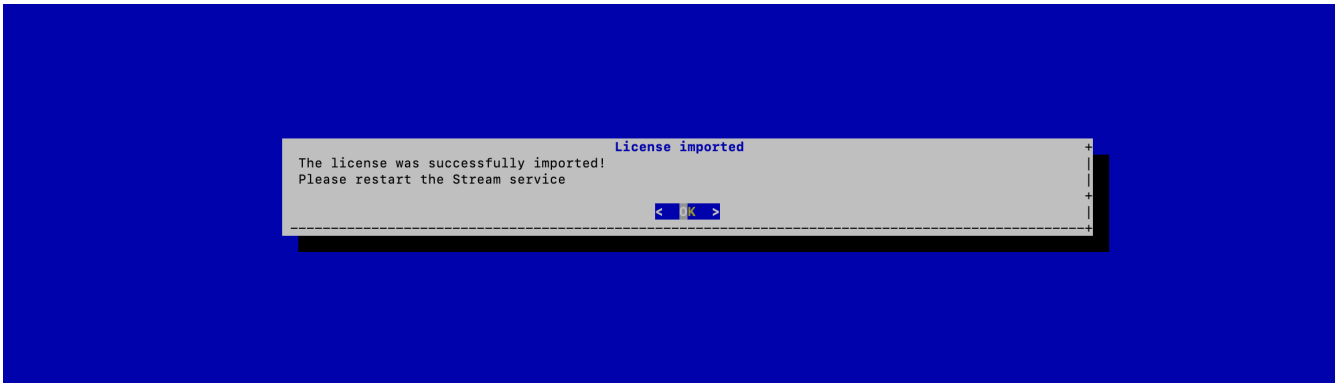
In the Stream configuration menu, Select **STREAM_LICENSE**:



Specify the path `/tmp/stream.lic` and validate:



The Stream License is configured:



For the changes to take effect, you must restart the Stream service by running:

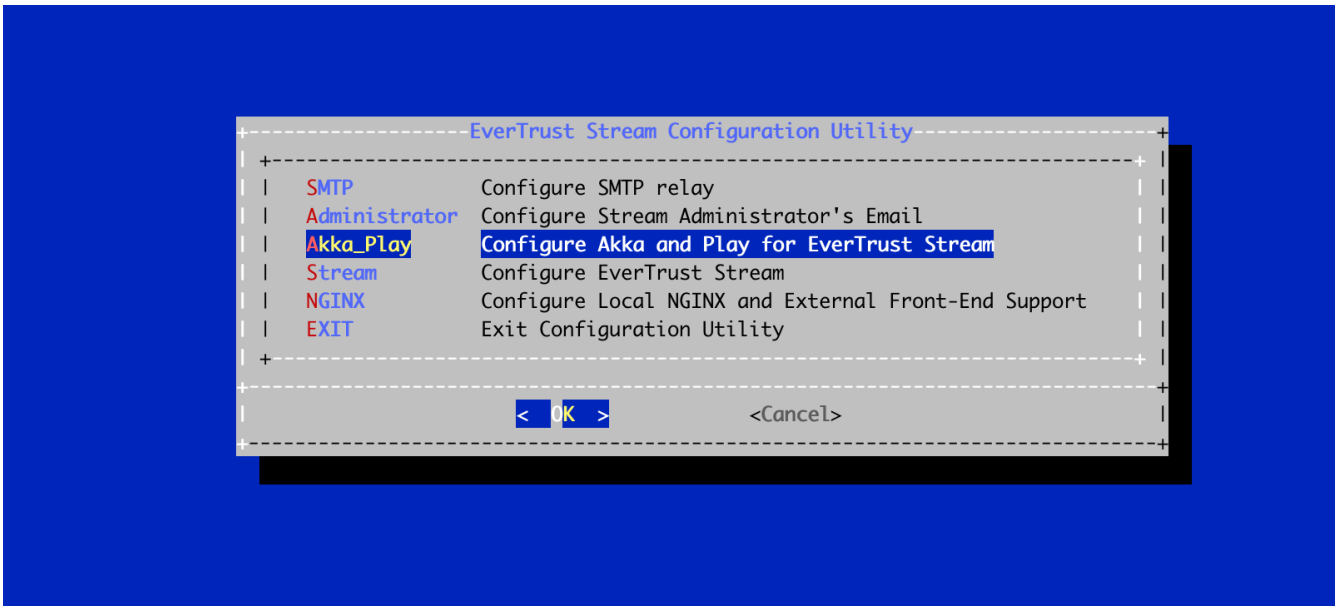
```
# systemctl restart stream
```

Installing Stream on a cluster of servers

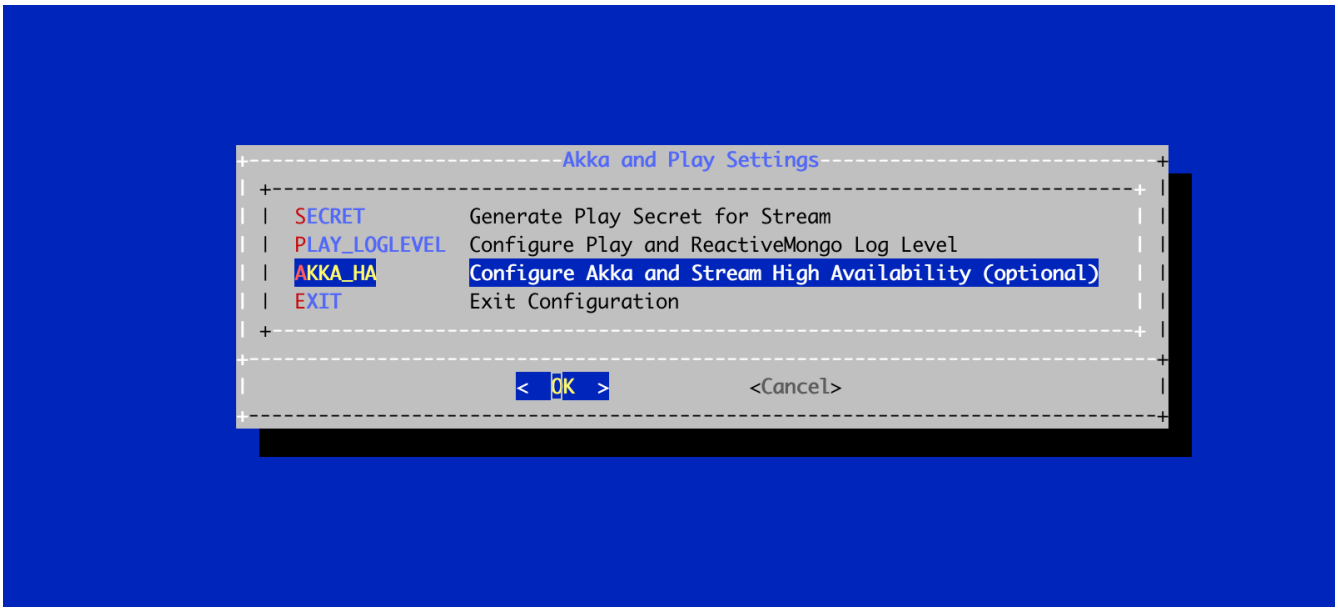


This section must not be followed if you plan on deploying Stream in standalone mode (vs cluster mode). WARNING: This section does not explain how to install Stream on a Kubernetes cluster. Please refer to the dedicated section.

In the main menu, select '**Pekko_Play**':



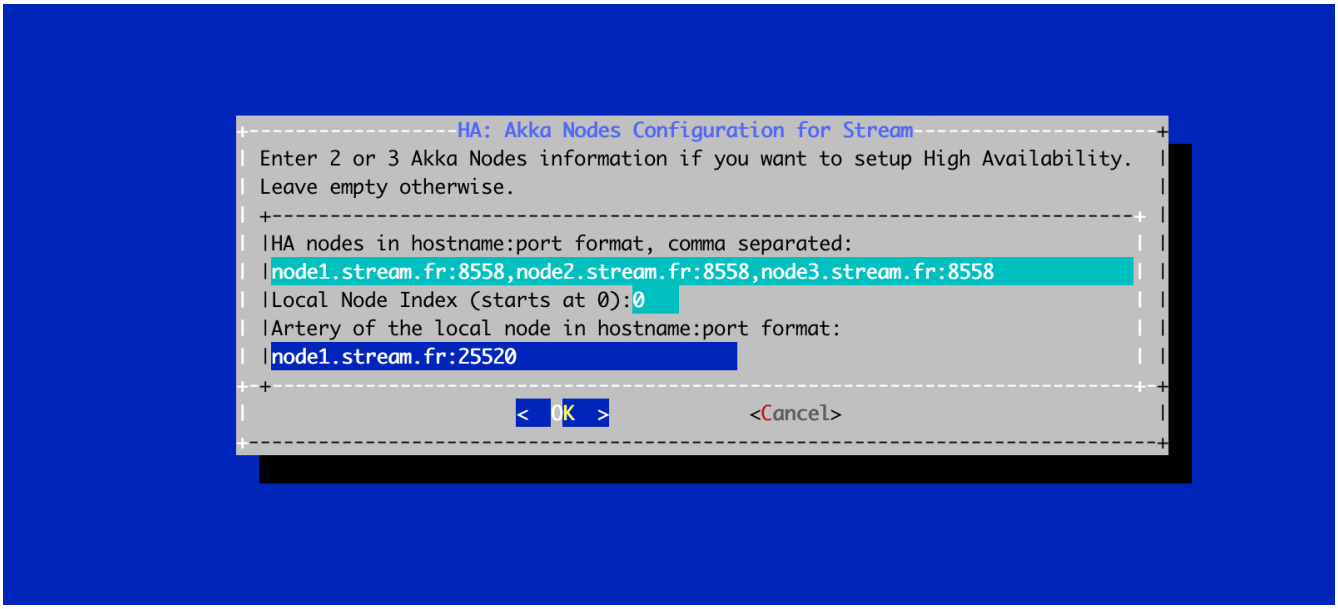
In the Pekko_Play menu, select '**PEKKO_HA**':



In this menu, specify either the IP address or the DNS name for each server that will be running Stream on this cluster with pekko management port, as well as the local node index (the number of the node that you are configuring at that moment).

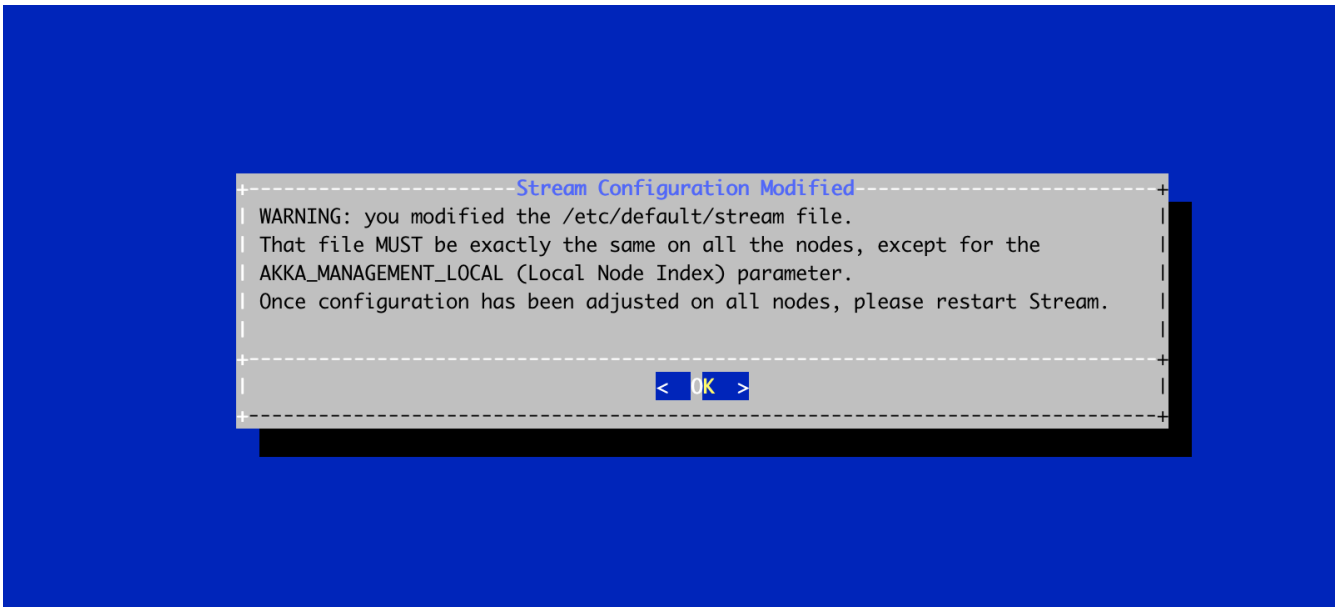


Note that the local node index must match the current node hostname or IP parameter:



Save your changes from the menu.

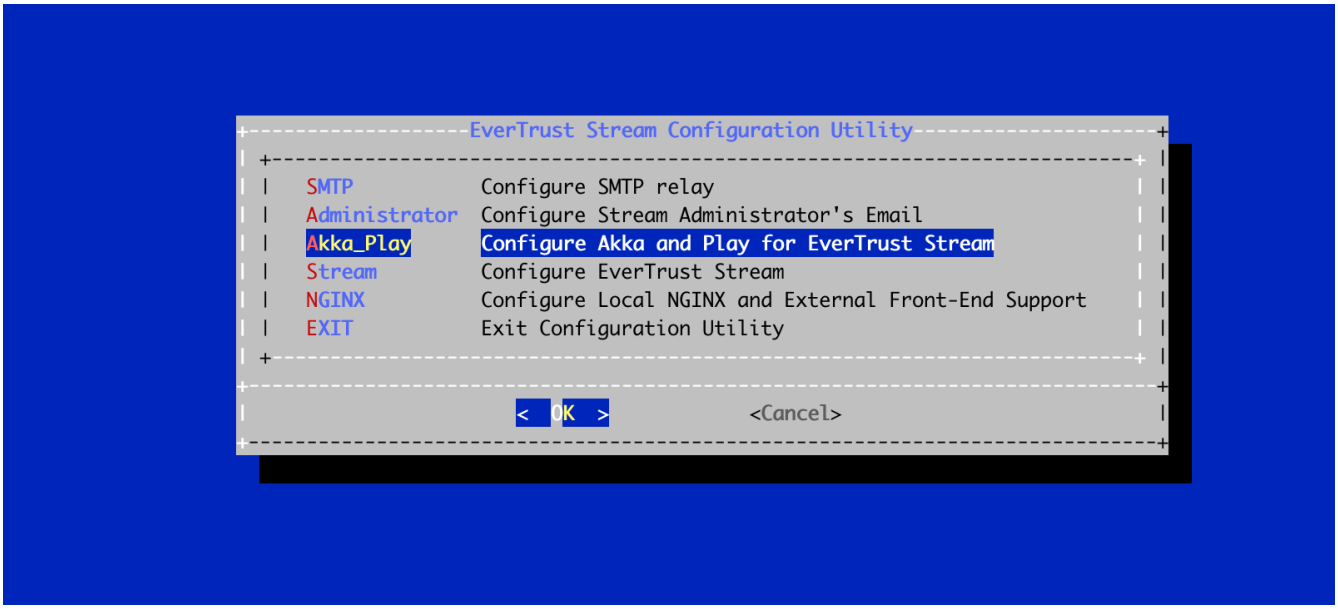
The High Availability mode is now configured on the current node:



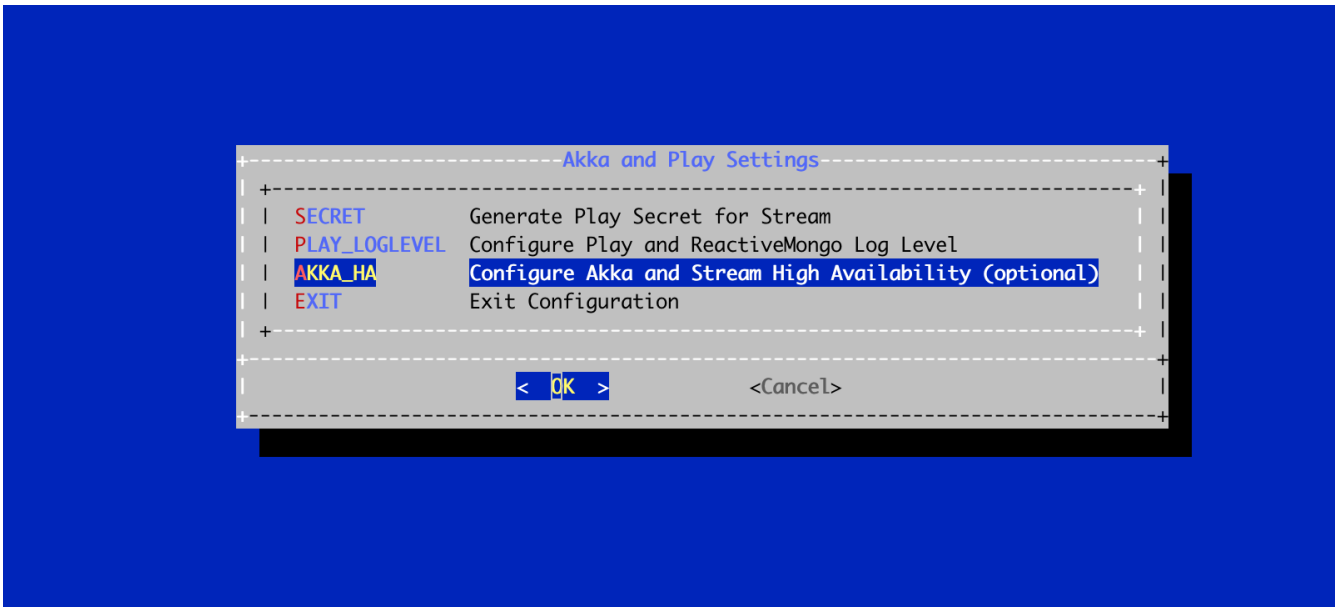
You must now configure your other nodes, but because they belong to the same cluster they need to share the **same pekko play secret, the same stream licence, the same stream seal secret, the same stream hostname, the same mongo database, the same x509 enforcing and the same stream tink keyset**. In order to be able to do that, you need to copy the configuration file that was generated by the stream-config app, named `/etc/default/stream` and paste it on each one of your nodes;

Then on each other node, run the Stream Configuration utility with the following command:

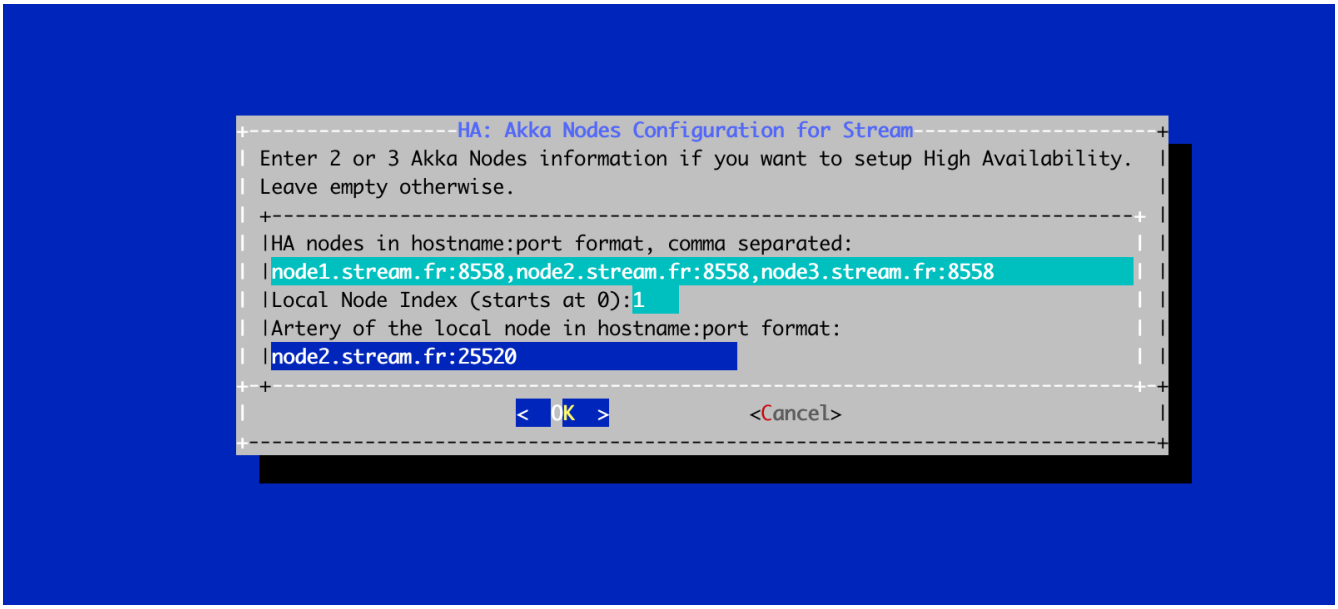
```
# /opt/stream/sbin/stream-config
```



In the Pekko_Play menu, select '**PEKKO_HA**':



Here, you need to change the local node index to match the hostname of the node that you are configuring:





You will need to import the Stream licence file on each node manually, following the guidelines of section Installing the Stream license.

Additionally, on each node, you will need to open the ports used for Pekko_HA and Pekko_MGMT, which are by default 25520 and 7626:

```
# firewall-cmd --permanent --add-port=25520/tcp
# firewall-cmd --permanent --add-port=7626/tcp
```

Reload the firewall configuration with:

```
# systemctl restart firewalld
```

Restart the Stream service on each one of the nodes:

```
# systemctl restart stream
```

Enabling the lease

To allow for High Availability even when a minority of nodes are up, the following configuration should be added (reference).

```
pekko.cluster.split-brain-resolver {
    active-strategy = "lease-majority"
    lease-majority {
      lease-implementation = "lease.mongo"
    }
}
```

Bootstrapping EverTrust Stream

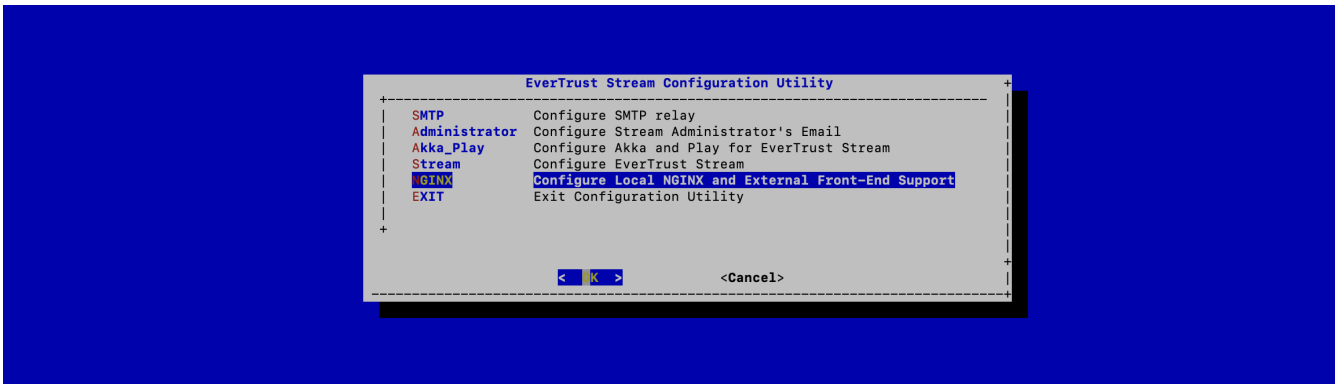
Installing a bootstrap certificate

Connect to the server with an account with administrative privileges;

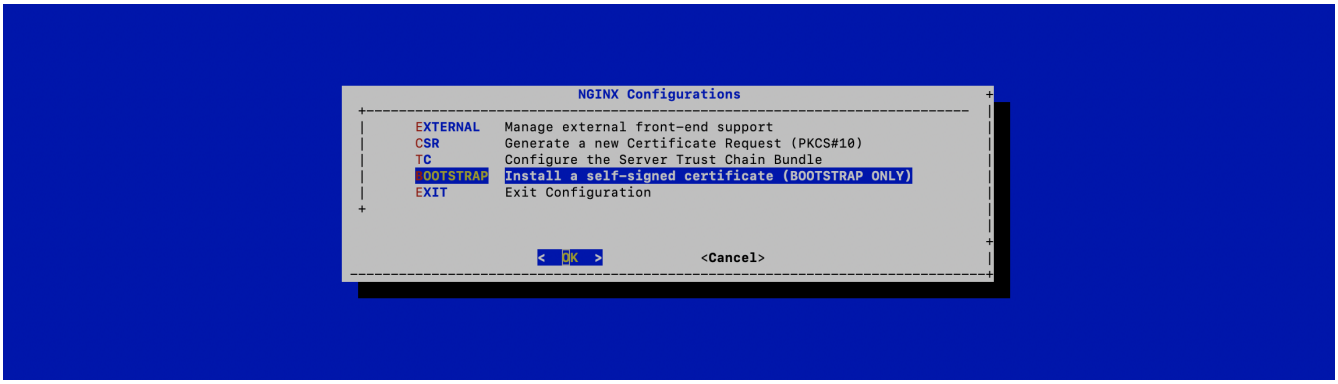
Start the Stream configuration utility by running:

```
# /opt/stream/sbin/stream-config
```

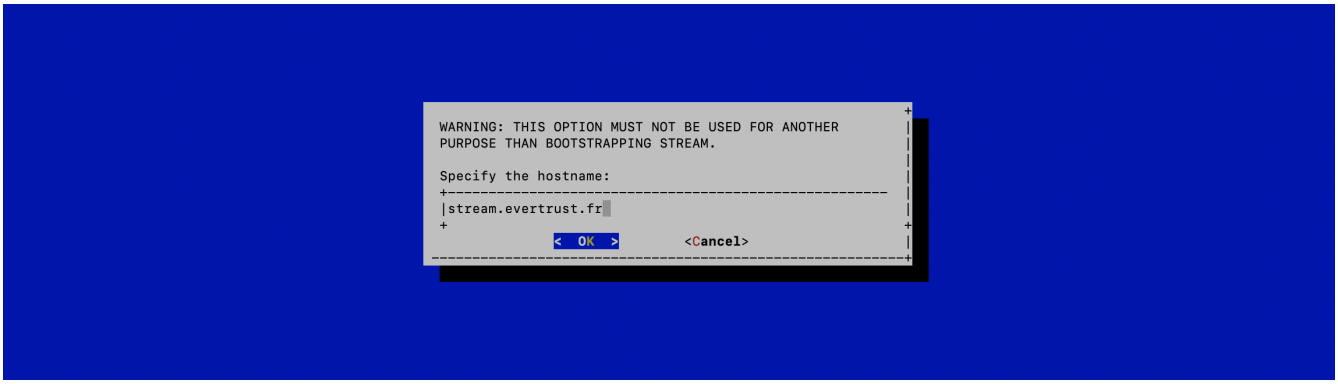
In the main menu, select '**NGINX**':



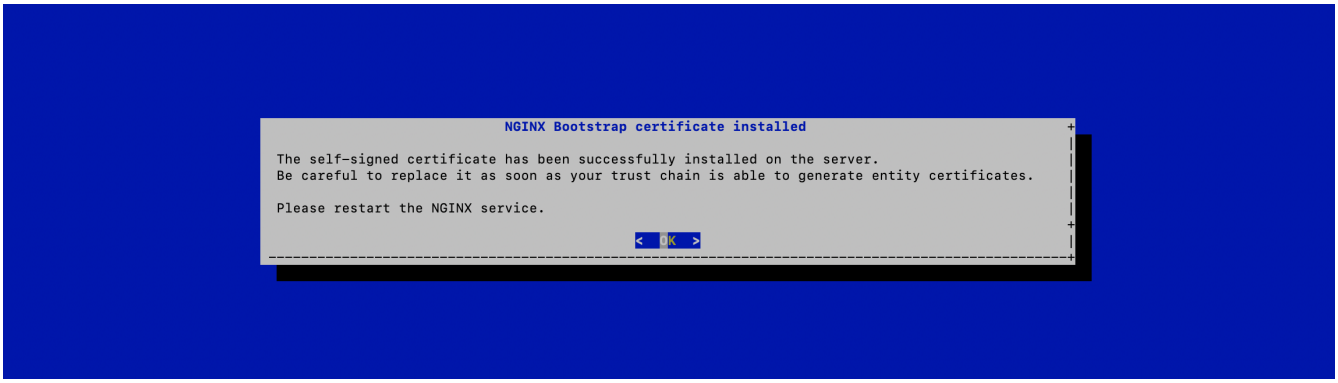
In the NGINX menu, select '**BOOTSTRAP**':



Specify the DNS Name of the Stream server (the same that you used as Stream hostname previously):



The self-signed certificate is going to be generated and automatically installed for Nginx to use directly:





This certificate is meant to be used to bootstrap Stream and should be replaced as quickly as possible as it is highly unsecured.

Starting the services

Assuming that all prior configuration operations have been performed as documented in the installation guide and that a bootstrap certificate has been installed as explained in the previous section, the services must be started:

1. Connect to the server with an account with administrative privileges;
2. Start the stream service with the following command:

```
# systemctl start stream
```

3. Verify the NGINX configuration with the following command:

```
# nginx -t
```

4. Restart the NGINX service with the following command:

```
# systemctl restart nginx
```

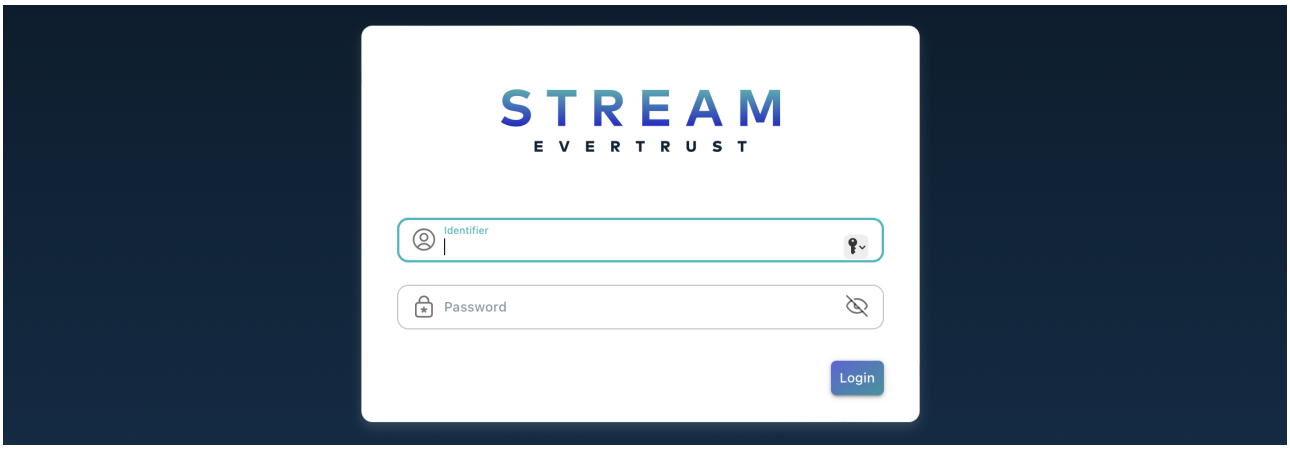
Retrieving initial password

For the first log-in, you must find the administrator password in the `/opt/stream/var/run/adminPassword` file.

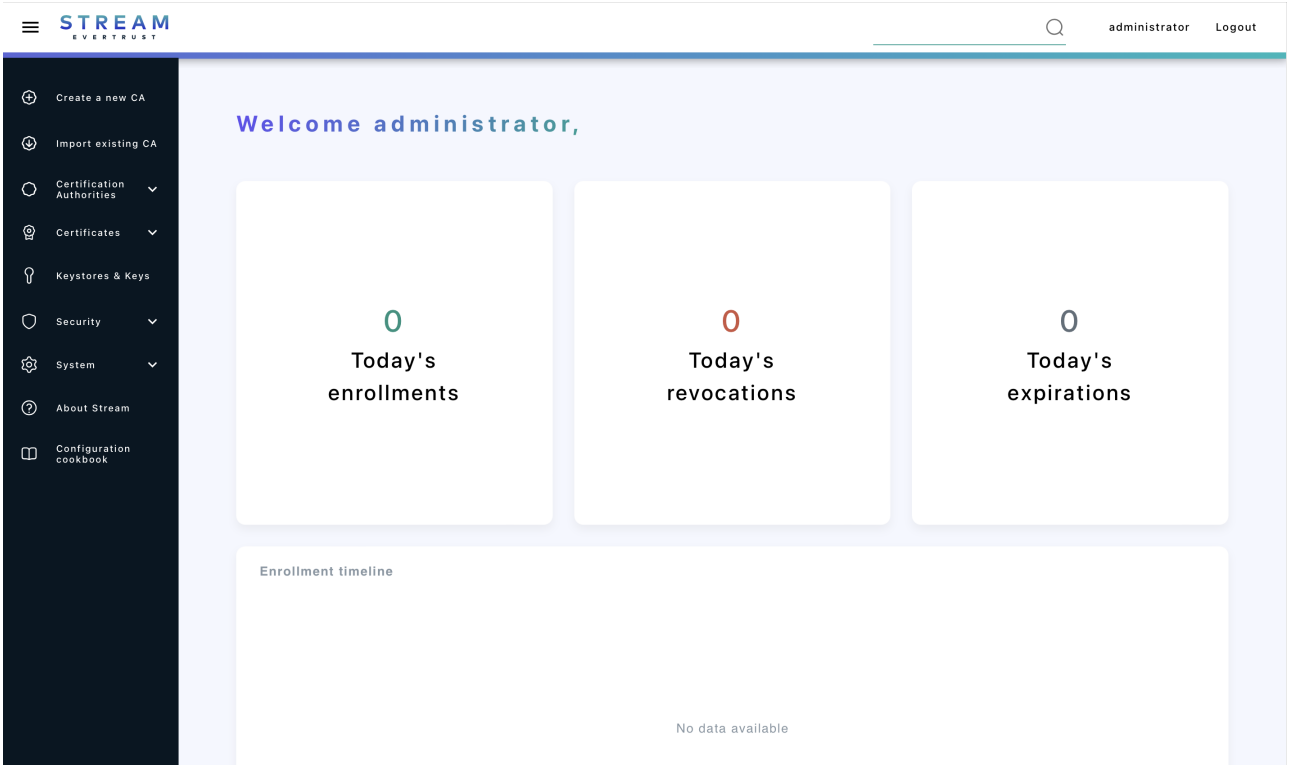
The default administration login is `administrator`.

Accessing the Stream Web Interface

1. Launch a web browser;
2. Browse to `https://[IP or DNS Name of the Stream component]/ui#`:
3. When prompted with a security issue, click on the button to accept the risks and proceed anyway. This alert is raised by the use of the self-signed certificate.



4. Specify the default administration credentials and hit the '**Login**' button:





It is **highly recommended** to delete the `adminPassword` file from your machine once you saved it somewhere safe.

Initial Key Ceremony

Before deploying to production, the initial key ceremony should take place

Configure a keystore

To protect the keys, keystores (cloud or physical) should be configured. Follow the Administration Guide steps in **Managing Keystores & Keys › Keystores in Stream** to configure your Keystore.

Create keys

A key should be created for each Certification you wish to add. The keys can be generated externally, or using Stream.

Key creation steps depend on the type of keystore:

- **KMS:**
  KMS keys can be created using Stream following the Administration Guide steps in **Managing Keystores & Keys › Managing keys in Stream › Cloud KMS** or directly in the KMS following your KMS documentation.
- **Software Keystore:**
  Software keys can be created using Stream following the Administration Guide steps in **Managing Keystores & Keys › Managing keys in Stream › Software keystore**.
- **Hardware Security Module:**
  HSM keys can be created using Stream following the Administration Guide steps in **Managing**

Keystores & Keys › Managing keys in Stream › PKCS#11 HSM. Please note that extra steps may be required at HSM level depending on the model of HSM used.

Once the keys have been created, they should appear in the keystore on Stream after a refresh.

Create your Certification Authorities

Once keys have been created the Certification Authorities can be created following the Administration Guide steps in **Managing Certification Authorities**.

Finalizing Stream Configuration

Configuring Stream default templates

If you intend to use Stream as your certification authority, default templates should be created.

As Stream is template-oriented, default templates will be used when enrolling certificates.

Two templates are considered as defaults:

- A `tlsClient` template for TLS client certificates
- A `tlsServer` template for TLS server certificates

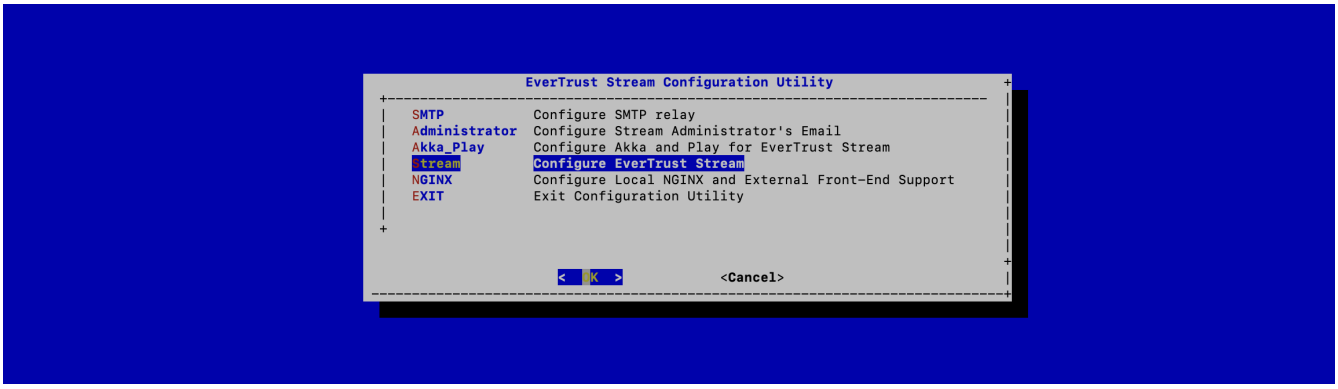
To create these templates:

Connect to the server with an account with administrative privileges;

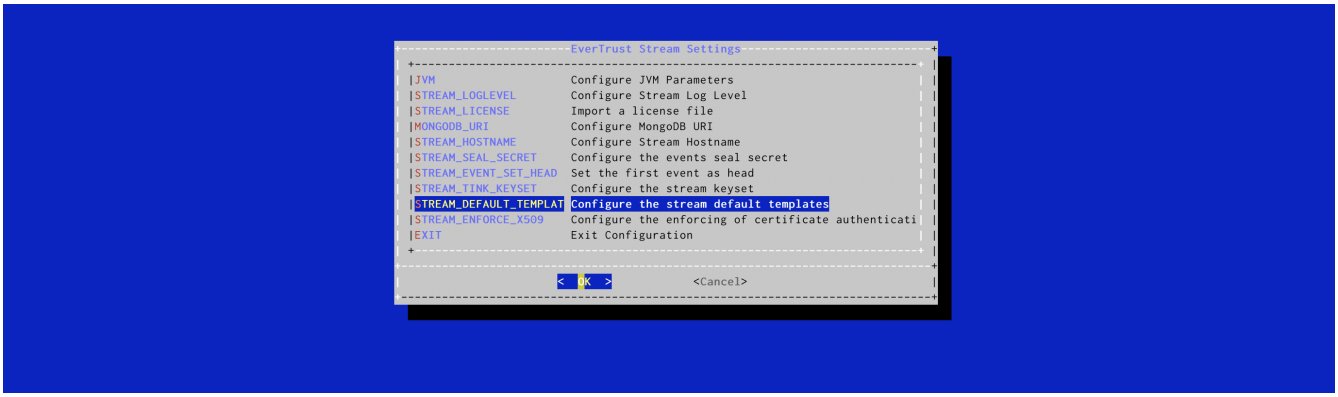
Start the Stream configuration utility by running:

```
# /opt/stream/sbin/stream-config
```

In the main menu, select '**Stream**':



In the Stream menu, select '**STREAM_DEFAULT_TEMPLATE**':



The templates are automatically created and available in **Certificates › Templates**.



`mongosh` must be installed and the mongo URI configured for this configuration to work

Installing a Server Authentication Certificate

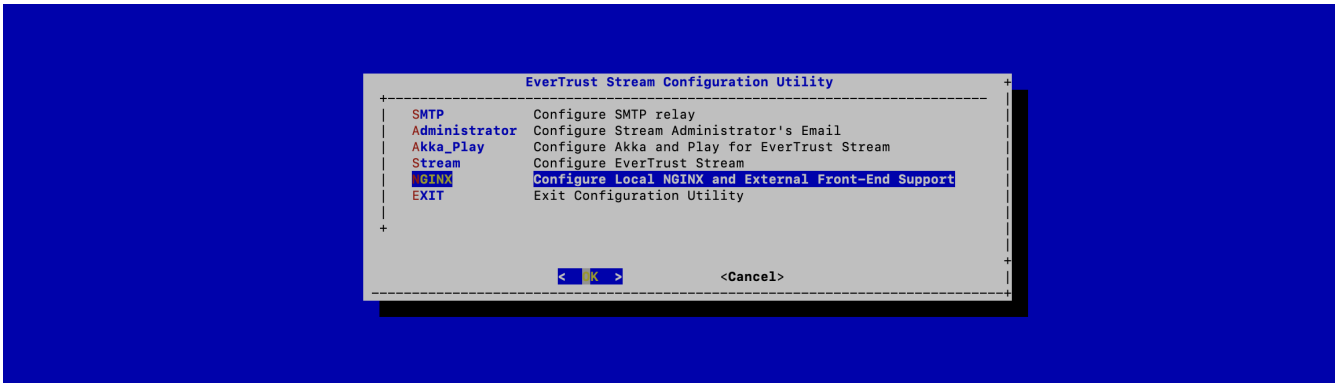
Issuing a Certificate Request (PKCS#10)

Connect to the server with an account with administrative privileges;

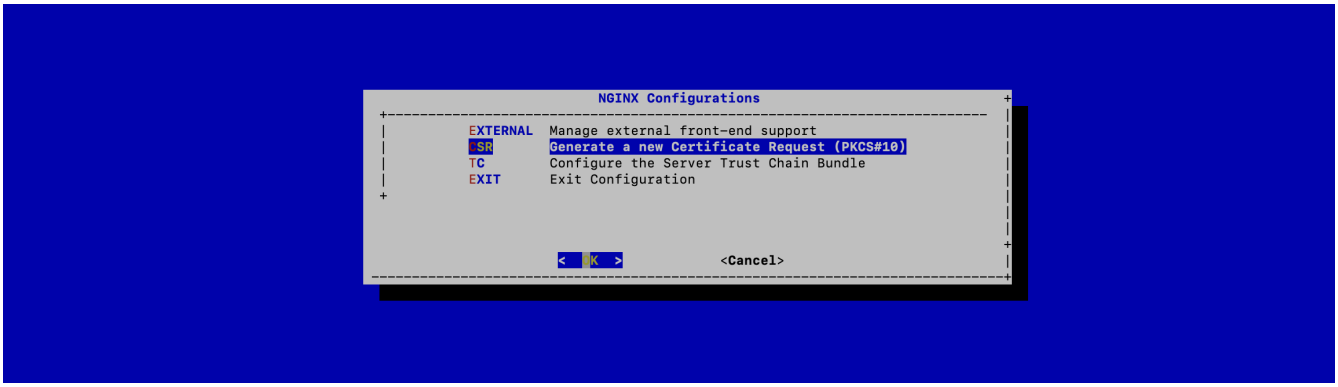
Start the Stream configuration utility by running:

```
# /opt/stream/sbin/stream-config
```

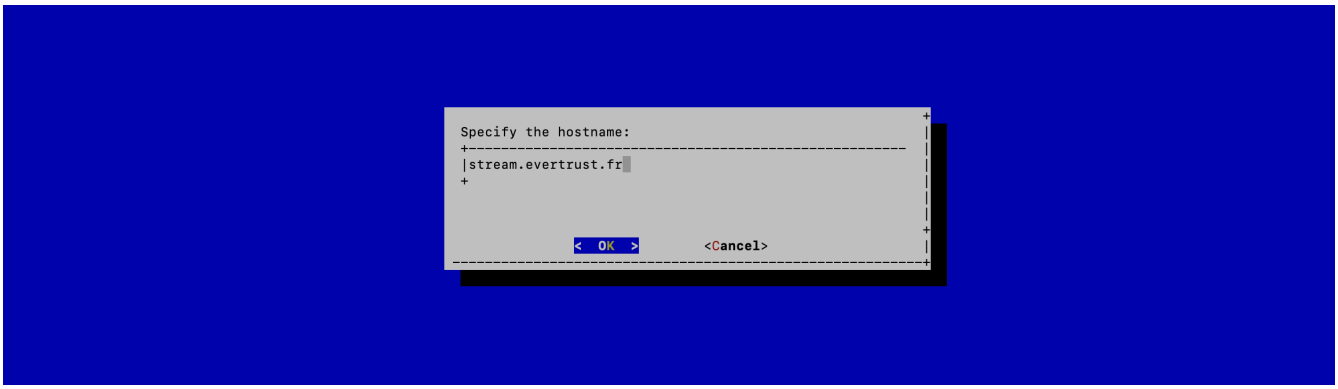
In the main menu, select '**NGINX**':



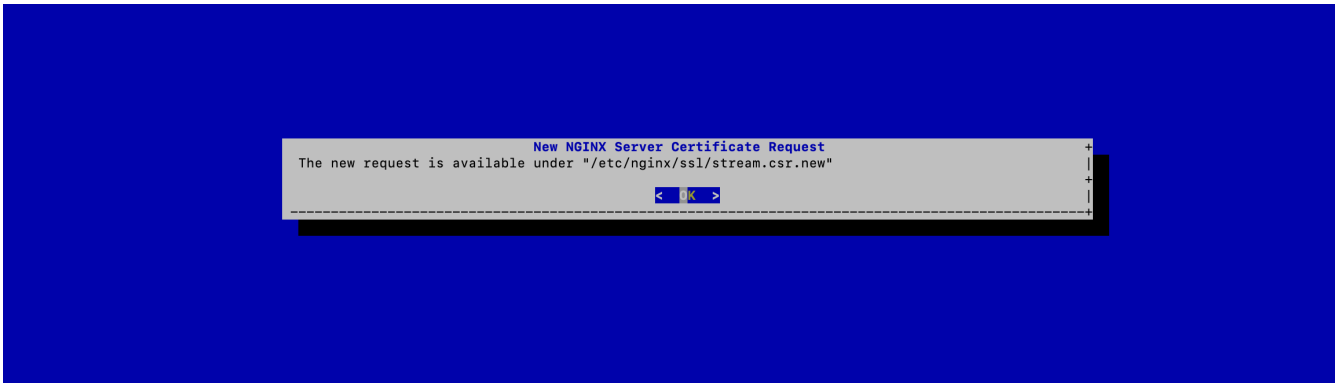
In the NGINX menu, select '**CSR**':



Specify the DNS Name of the Stream server (the same that you used as Stream hostname previously):



The certificate request is generated and available under '***/etc/nginx/ssl/stream.csr.new***':



Signing the certificate

The CSR generated at the previous steps must then be signed using an existing PKI. If you do not have an existing PKI, please refer to the Key Ceremony Documentation to create one.

Sign using Stream



This step assumes that the Key Ceremony already happened, i.e the operational CAs are imported into Stream as managed CAs.

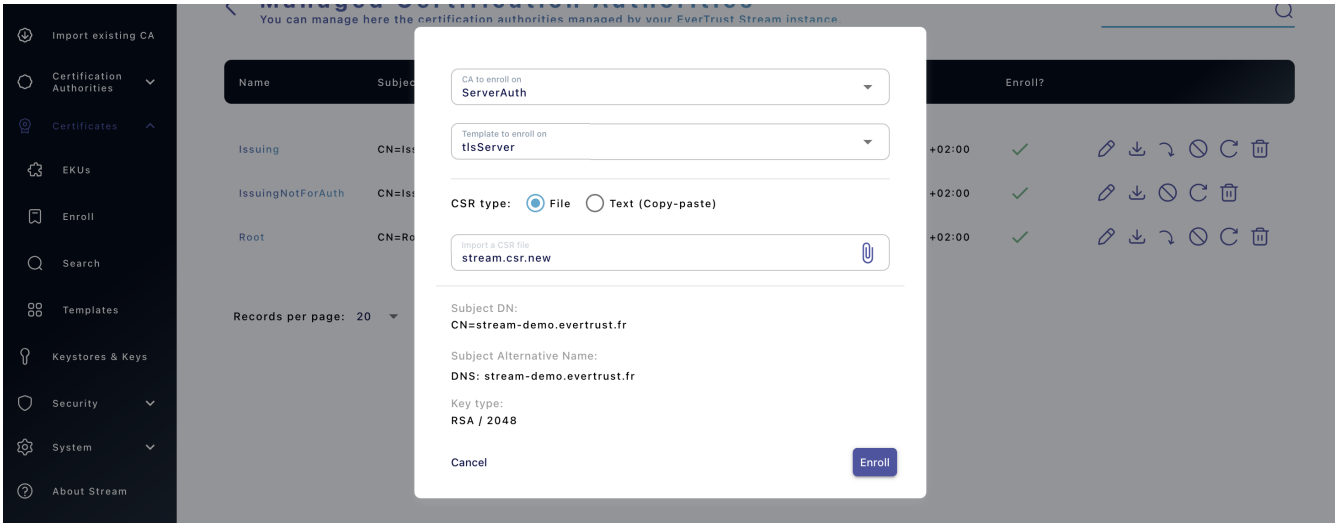
1. Access Stream's web administration console and log in using the default administration credentials that were obtained at this section;

2. Browse to '**Certificates** > **Enroll**';

3. Fill in the following information:

- CA to enroll on: Select the CA that is issuing the server certificates for your organization;
- Template to enroll on: Select the certificate template that was created at this step that should be named 'tlsServer';
- CSR type: Select 'File', then click the paper clip icon and import the CSR that was generated at this step;

Then click 'Enroll':



You can retrieve the enrolled certificate in PEM format from the '**Certificates** > **Search**' : download this certificate in PEM

Sign using an external Certification Authority



The certificate must have the `serverAuthentication` Extended Key Usage

You will need to provide your certificate authority with the `/etc/nginx/ssl/stream.csr.new` file that was generated at the previous step.

Installing the Server Certificate

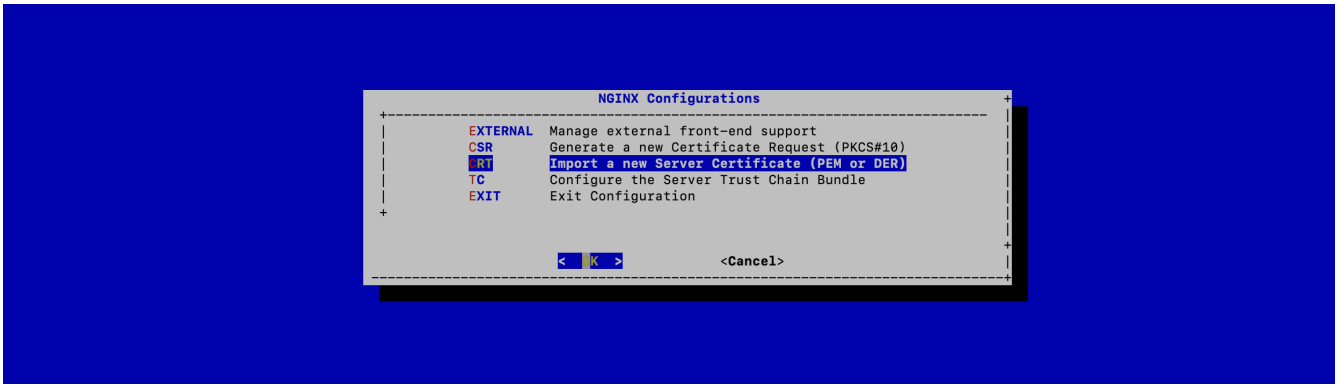
Upload the signed server certificate (in PEM format) on the Stream server under `/tmp/stream.crt` (using SCP or other means);

Connect to the server with an account with administrative privileges;

Start the Stream configuration utility by running:

```
# /opt/stream/sbin/stream-config
```

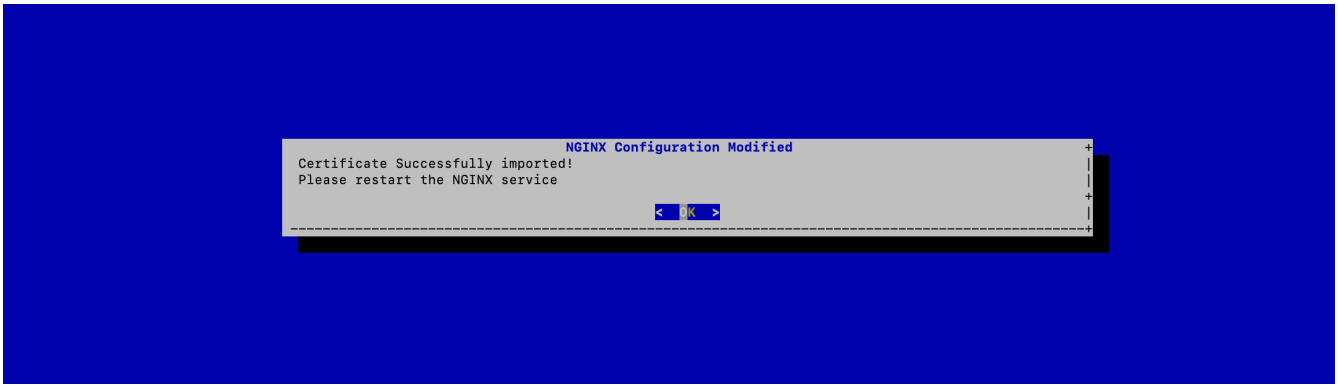
In the NGINX configuration menu, select '**CRT**':



Specify the path `/tmp/stream.crt` and validate:



The server certificate is successfully installed:



Installing the Server Certificate Trust Chain



You must follow this section only if you signed the server certificate with an existing PKI. If you self-signed the server certificate, you do not need to follow this step.

Upload the server certificate trust chain (the concatenation of the Certificate Authority certificates in PEM format) on the Stream server under `/tmp/stream.bundle` (using SCP or other means);



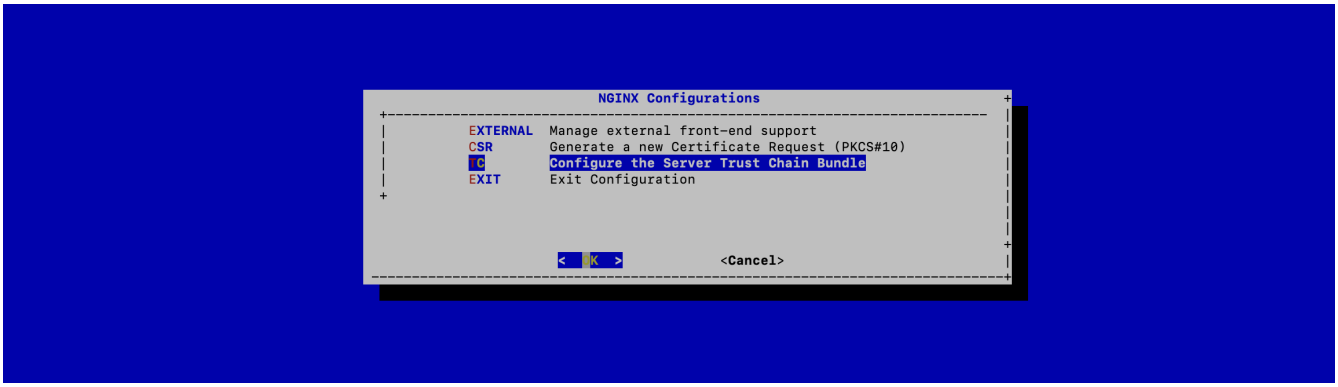
The bundle should contain only the Certificate Authority certificates in PEM format and **NOT** the server certificate

Connect to the server with an account with administrative privileges;

Start the Stream configuration utility by running:

```
# /opt/stream/sbin/stream-config
```

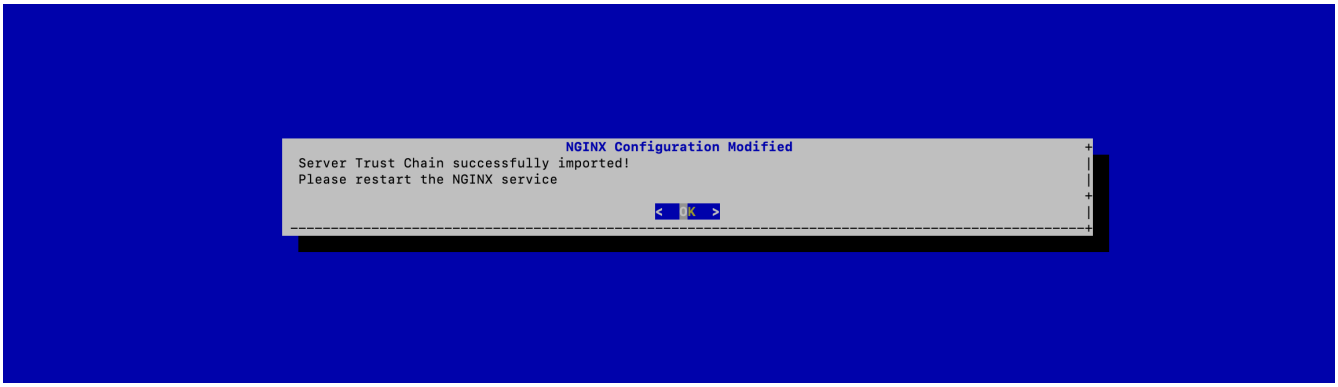
In the NGINX configuration menu, select '**TC**':



Specify the path `/tmp/stream.bundle` and validate:



The server bundle is successfully installed:


NGINX Configuration Modified
Server Trust Chain successfully imported!
Please restart the NGINX service
< OK >

Verify the NGINX configuration with the following command:

```
# nginx -t
```

Restart the NGINX service with the following command:

```
# systemctl restart nginx
```

Configuring administrator certificate authentication

To authenticate as administrator on stream using a certificate, it is possible to either use a certificate generated on Stream, or to use an external certificate.

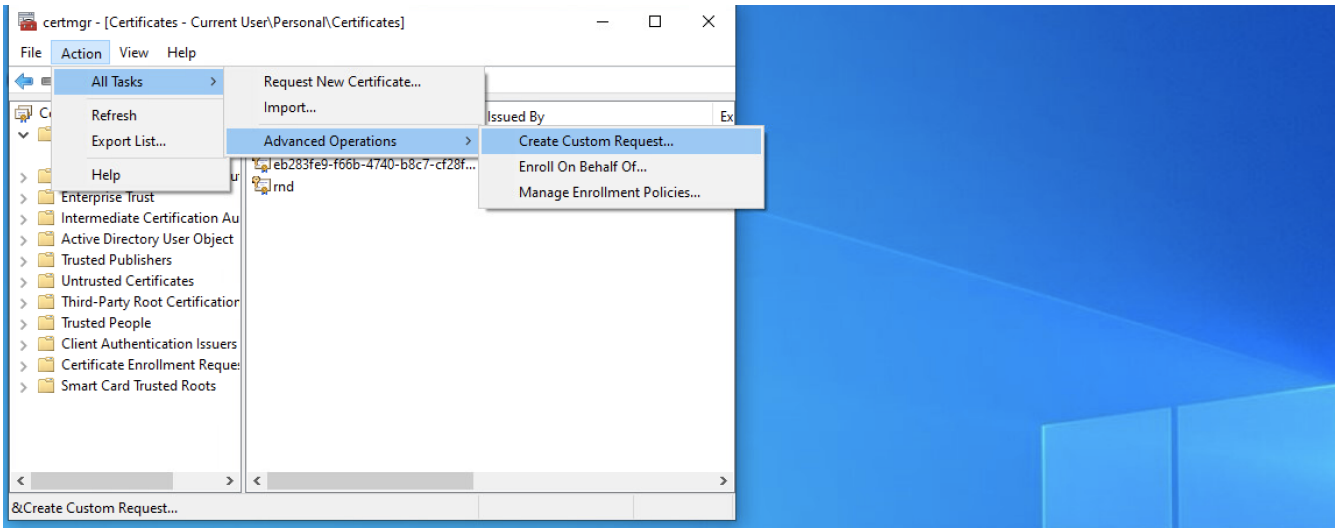
Issuing a Certificate Request (PKCS#10)

Generating the CSR on Windows

1. Press 'Windows Key + R' then type 'certmgr.msc' and press enter;

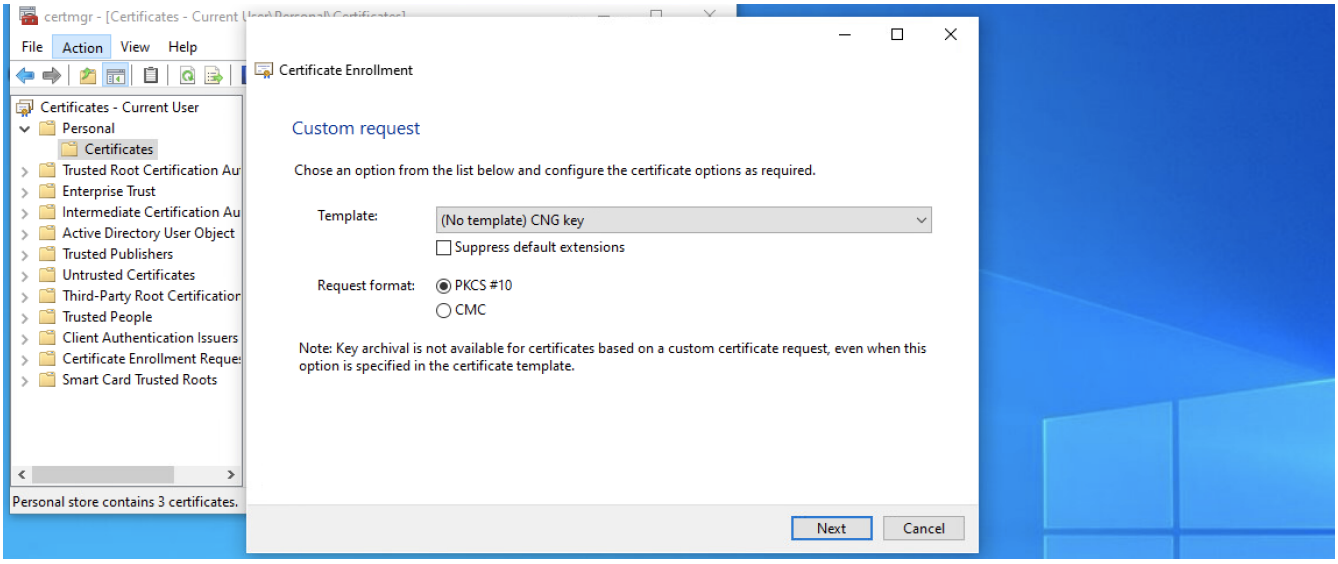
2. Expand 'Personal' then double click 'Certificates';

3. In the top bar, click 'Action > All Tasks > Advanced Operations > Create custom requests':

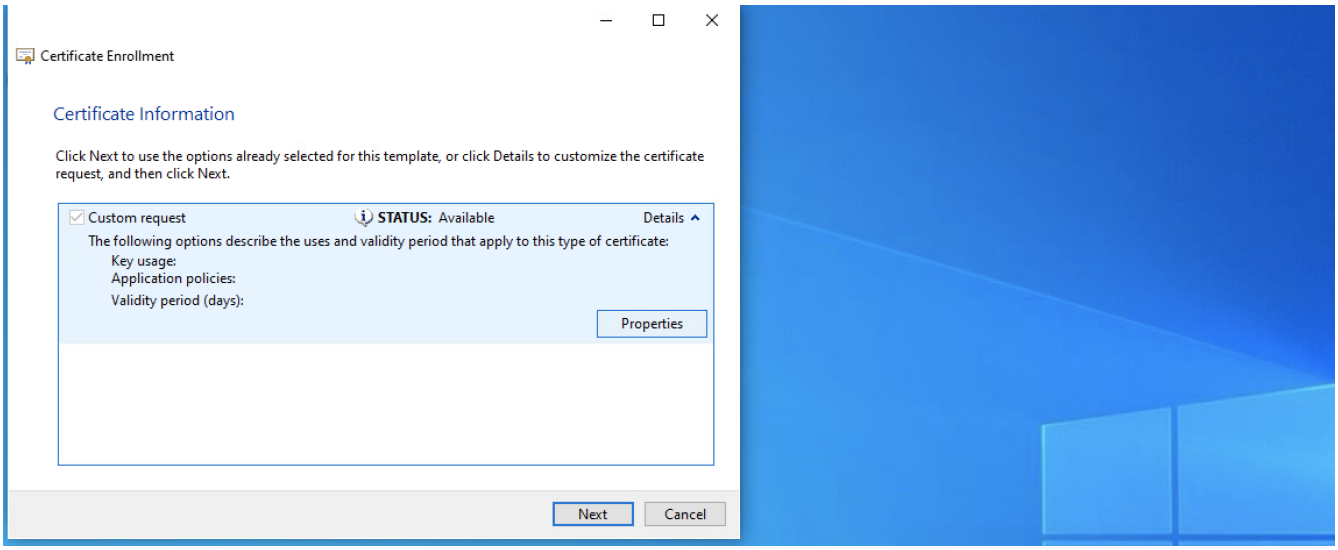


4. On the 'Before You Begin' and the 'Select Certificate Enrollment Policy' screens, click 'Next';

5. In the 'Custom request' window, select 'CNG Key' in the template drop-down menu and 'PKCS#10' as the request format, then click 'Next':



6. In the 'Certificate Information' window, click 'Details' then 'Properties':

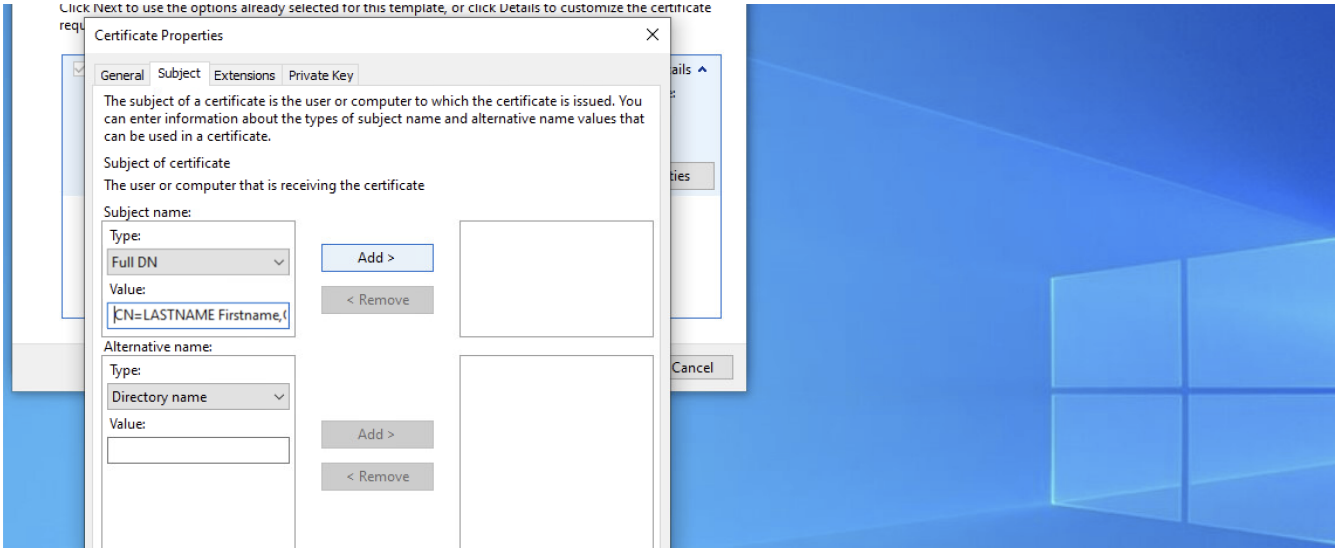


7. In the 'Certificate Properties' pop-up that opened:

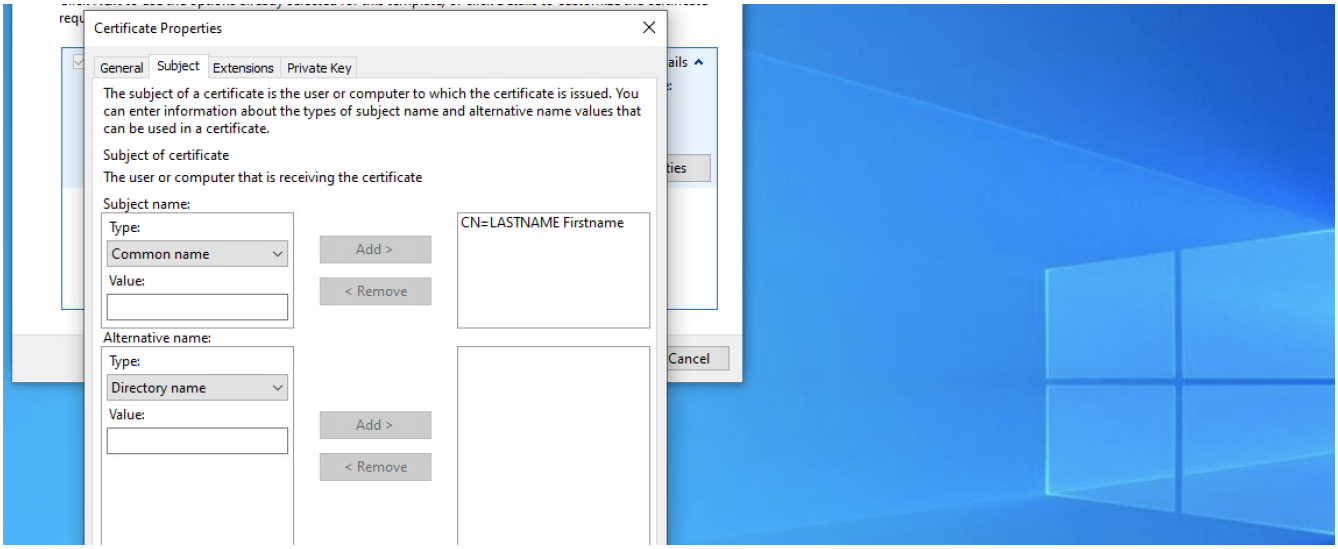
- General tab: Leave everything as default
- Subject tab: In the subject name category enter the DN using the standard DN format then click Add:



It is recommended for a certificate DN to contain at least a CN, OU, O and C element

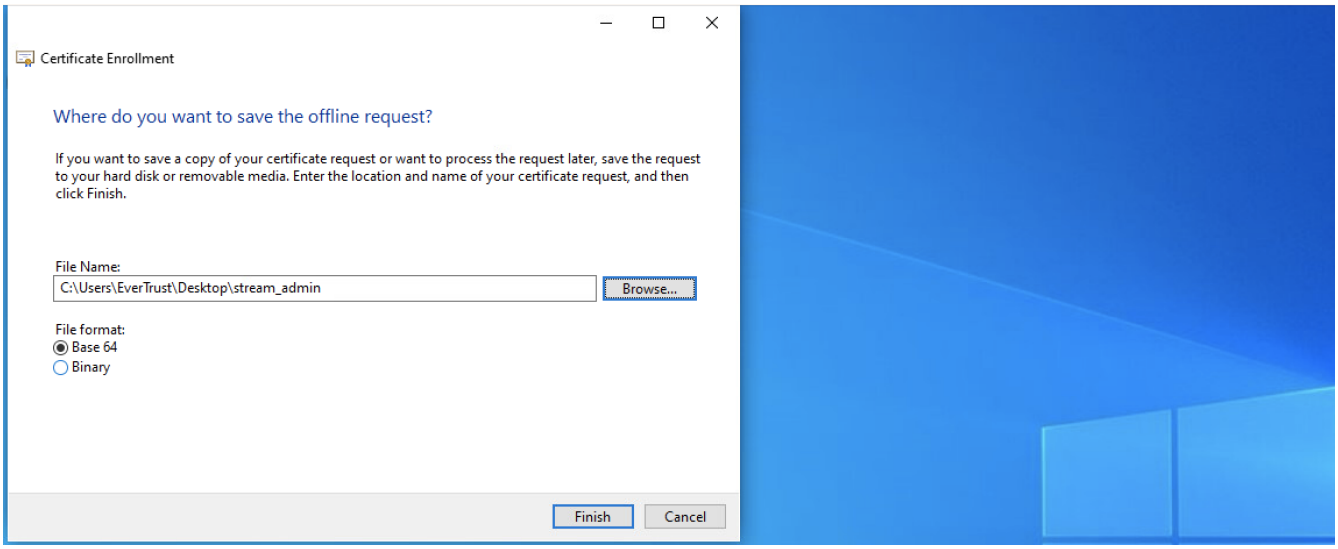


- Extensions tab: Leave everything as default
- Private Key tab: Expand 'Key options' and select '2048' for the key size, then expand 'Select Hash Algorithm' and select 'sha256' for the hash algorithm:



Click apply, then OK, then Next;

8. In the next window, select a place to save the CSR and select 'Base64' as File format, then click 'Finish':



Generating the CSR on Linux/MacOS

1. Start a shell instance;

2. Run the following command, replacing the DN with your information:

```
$ openssl req -new -newkey rsa:2048 -subj "/CN=LASTNAME Firstname/C=FR/OU=.../"
-keyout ./stream_admin.key -out ./stream_admin.csr
```

This command will prompt you for a password to encrypt the CSR private Key



It is recommended for a certificate DN to contain at least a CN, OU, O and C element

Signing the certificate

Using a Stream-signed certificate



This step assumes that the Key Ceremony already happened, i.e the operational CAs are imported into Stream as managed CAs.

1. Access Stream's web administration console and log in using the default administration credentials that were obtained at this section;

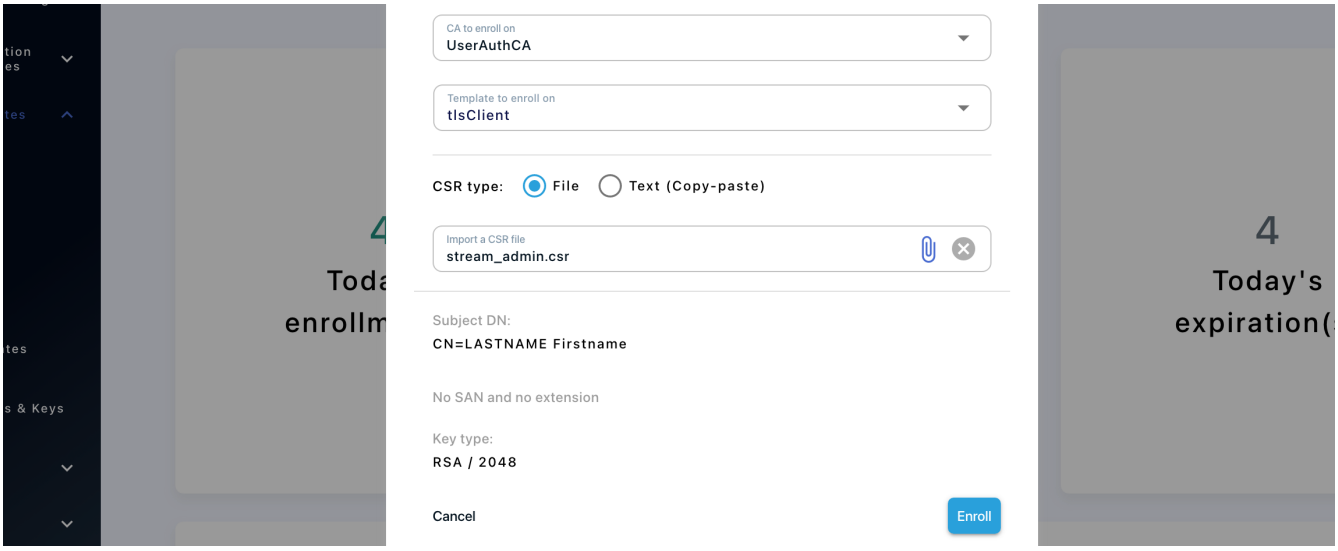
2. Browse to '**Certificates** > **Enroll**';

3. Fill in the following information:

- CA to enroll on: Select the CA that is issuing the user certificates for your organization. Note that it must have the 'Is trusted for client authentication' switch turned on in the interface;
- Template to enroll on: Select the certificate template that was created at this step that should be named 'tlsClient';
- CSR type: Select 'File', then click the paper clip icon and import the CSR that was generated at

this step;

Then click 'Enroll':



You can retrieve the enrolled certificate in PEM format from the '**Certificates** > **Search**' : download this certificate in PEM

Using an external-signed certificate



The certificate must contain the clientAuthentication Extended Key Usage in order to be used as an authentication certificate on Stream

Trusting the issuing CA for client authentication

In order for the client authentication certificate to work as intended, the issuing CA must be trusted for client authentication on Stream.

- If your certificate was signed by a Stream managed CA:
  Browse to **Certification Authorities › Managed CAs**, select your Issuing CA and toggle on '**Is trusted for client authentication**'
- If your certificate was signed by an external CA:
  If your CA is not yet imported, import it using the administration guide steps in **Managing Certification Authorities › Importing an External Certification Authority**.
  Browse to **Certification Authorities › External CAs**, select your Issuing CA and toggle on '**Is trusted for client authentication**'

Creating the SuperAdmin role in Stream

At this point, you should be logged into the Stream web administration console with the default administrator account.

1. Go to **Security** > **Roles** and click ⊕ ;

2. Fill-in the following information:

- **Name**: SuperAdmin;
- **Description** : Super administrator role that has all rights on Stream. Use with caution;
- **Configuration permissions** : Click the '+' button then click on 'Section' and select 'All permissions of all type' and click the 'Add' button;
- **Lifecycle permissions** : Click the '+' button, then :
  - Click on 'CA' and select 'All';
  - Click on 'Template' and select 'All';
  - Click on 'Rights' and select 'All';
  - Click the 'Add' button.

3. Click the '**Save**' button at the top of the page.

The SuperAdmin role is now created.

Giving the SuperAdministrator role to the certificate

1. Access Stream's web administration console and log in using the default administration credentials that were obtained at this section;

2. Browse to '**Security** > **Authorizations**' then click  and select 'Add authorization manually';

3. Import a certificate by clicking on certificate button ;

4. Once the identifier has been filled with the certificate DN click the 'Add' button;

4. Click the 'Add role' button and select the 'SuperAdmin' role previously created from the drop down menu;

5. Click the 'Save' button at the top of the page;

Now the PEM certificate has the SuperAdmin permission in Stream.

Creating a PKCS#12 file from the private key and the PEM

This step can only be done using OpenSSL.

1. Connect to the server with an account with administrative privileges;

2. Upload the PEM of the certificate you used to create the SuperAdmin user at the previous step in the same folder where the private key that was used to generate the CSR was created in;

3. Run the following command:

```
$ openssl pkcs12 -export -inkey ./stream_admin.key -in ./stream_admin.pem -name
StreamAdministrator -out ./stream_admin.p12
```

This will create a PKCS#12 file containing the PEM of the certificate and its private key in PKCS#8

format (encrypted). The previous utility will first ask the private key file pass phrase and then ask for a passphrase to protect the PKCS#12.



If using an openssl version greater than 3, you will need to add `-legacy` at the end of the command as the new version of the PKCS#12 is not wisely supported.

4. Import this .p12 file into your certificate store or the certificate store of your browser if it has one (ex: Firefox). At the import, you should be prompted to enter the passphrase that you used to encrypt the PKCS#12.

Now you should have the certificate imported and ready to be used.

When going to the Stream web interface while not being logged in, your web browser should prompt you for a certificate to use and you should be able to select the one that you imported just now.

Removing the administrator local account



Before deleting the account, ensure that your administrator certificate is working as intended.

1. Access Stream's web administration console and log in using the client certificate that was set-up at the previous step;

2. Browse to '**Security** > **Local accounts**';

3. Click the bin icon next to the '*administrator*' account and confirm deletion.



After reviewing the [install-guide:iaas-security:::_security_guidelines], Stream is ready to leave its confined environment.

1.2.4. Security Guidelines

The following content are guidelines to have a secure Stream installation.

Stream should run on a dedicated machine.From this fact, all unused packages should be removed from the machine.The system should have been installed following the security guidelines recommended by the operating system vendor.

The following requirements should be met:

- SELinux should be enabled
- The `stream-hardening` rpm should be installed
- Privileged Access Management or SSH/Sudoers should be set up
- The firewall should be enabled and ports 80 and 443 allowed
- Only certificate authentication should be enabled once the product has been initialized and the initial Key Ceremony phase performed

- The NGINX configuration should be modified following the below procedure
- Stacktrace logging in events should be disabled

On top of that, though it is not mandatory, it is recommended to set up other security-related solutions, such as a Web Application Firewall, an Intrusion Detection System and a Security Information and Event Management.

All the following steps should be followed to ensure compliance if they are not already implemented with the above requirements, and should be done with an account with administrative privileges.

SELinux

To enhance security, SELinux should be enabled.

```
# setenforce Enforcing
```

To ensure that it is enabled, run the following command

```
# getenforce
```

This should return `Enforcing`

Install the stream-hardening rpm

Follow the same steps as in Installing Stream but for the `stream-hardening` rpm.

Once the rpm is installed, a system reboot is necessary. The following command can be used:

```
# reboot now
```



In order to install the stream hardening policies, the server must have access to a repository (mirror, ISO file, ...) of the linux distribution you are using in order to be able to install the dependencies of the software. The `stream-hardening` package has the following dependencies:

- `stream`
- `policycoreutils-python-utils`

Please note that these packages may have their own dependencies.

Sudoers

To administrate Stream without using the root user, `stream-hardening` rpm creates a `stream-administrator` group with sudoers permissions.

Create a new user with `stream-administrator` and `stream` groups, for instance, user-admin.

```
# useradd -G stream-administrator,stream user-admin
# passwd user-admin
```

Link user-admin to the selinux `sysadm_u` user

```
# semanage login -a -s sysadm_u -rs0:c0.c1023 user-admin
```



The semanage command is available with the `policycoreutils-python-utils` package.

Relabel the user-admin user home folder with the following command

```
# restorecon -FR -v /home/user-admin
```

In case you need to access the user-admin user account via ssh you will need to set the selinux **ssh_sysadm_login** boolean

```
# setsebool -P ssh_sysadm_login on
```



The setsebool command is available with the `policycoreutils` package.

Now the user-admin can:

- Manage MongoDB server with systemctl
- Manage nginx server with systemctl
- Manage stream server with systemctl
- Execute every script under the folder `/opt/stream/sbin/` as a root user

Configuring the Firewall

The firewall should have been configured at the setup step.

In addition to this configuration, the https (443) access should be restricted to :

- The Stream administrators
- External components using Stream certificate lifecycle capabilities (Stream for example)



Firewalld sometimes has default ports allowed. No other ports than those referenced in the setup step should be allowed.

X509 Enforcing

In order to improve security once an administration certificate has been emitted, all authentication modes should be disabled apart from certificate authentication.

To do that, please follow the dedicated steps in the security section of the administration guide : **Managing Security › Enforce Certificate Authentication**

NGINX Configuration

In order to improve security, the default NGINX configuration should be altered. In the configuration file in `/etc/nginx/nginx.conf`, the `server` instruction block containing the `listen 80` instruction should be deleted or commented.

Once an administration certificate has been emitted, the `/opt/stream/etc/stream-httpd.conf` should be updated. The following line

```
ssl_verify_client         optional_no_ca;
```

should be replaced by the lines

```
ssl_verify_client         on;
ssl_client_certificate    ssl/client-trusted-cas.pem;
ssl_trusted_certificate   ssl/trusted-cas.pem;
ssl_crl                   ssl/crl-bundle.pem;
```

This ensures only valid and trusted certificates can be used to authenticate on the Stream server.

For the following example chain :
Root CA → Client Issuing CA 1
Root CA → Client Issuing CA 2

The `/etc/nginx/ssl/client-trusted-cas.pem` file should contain the PEM certificates of the CAs trusted for client authentication, concatenated one after the other. It should look like:

```
-----BEGIN CERTIFICATE----
<Client Issuing CA 1 PEM>
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE----
<Client Issuing CA 2 PEM>
-----END CERTIFICATE-----
```

The `/etc/nginx/ssl/trusted-cas.pem` file should contain the PEM certificates of the **chain** to the CAs trusted for client authentication, concatenated one after the other. It should look like:

```
-----BEGIN CERTIFICATE----
```

```
<Root CA PEM>
-----END CERTIFICATE-----
```

Both these files should have the following permissions:

```
# chmod 640 /etc/nginx/ssl/trusted-cas.pem
# chmod 640 /etc/nginx/ssl/client-trusted-cas.pem
# chown root:nginx /etc/nginx/ssl/trusted-cas.pem
# chown root:nginx /etc/nginx/ssl/client-trusted-cas.pem
```

To update the CRL, the `/opt/stream/etc/crl-fetching.conf` configuration file (installed with the `stream-hardening` rpm) should be customized

```
# Uncomment and edit the following line
# CRL_URLS=("<CRL URL 1>" "<CRL URL 2>")
CRL_DOWNLOAD_PATH="/etc/nginx/ssl/tmp.crl"
CRL_PEM_PATH="/etc/nginx/ssl/tmp.crl.pem"
TMP_CRL_BUNDLE_PATH="/etc/nginx/ssl/tmp.crl.bundle"
NGINX_CRL_BUNDLE_PATH="/etc/nginx/ssl/crl-bundle.pem"
```

To customize this file, after the `# Uncomment and edit the following line` comment, the `CRL_URLS` line should be uncommented and edited to have each of your CRLs URLs.



ALL of your CAs present in the `/etc/nginx/ssl/trusted-cas.pem` and `/etc/nginx/ssl/client-trusted-cas.pem` files must have their CRL downloaded.



CRL are expected in DER format.



To fetch the CRL on stream first follow **Managing Certificate Revocation Lists › Configuring Certificate Revocation Lists for a Managed CA** in the administration guide, the CRL path is `http://localhost:9000/crls/<your CA Technical Name>`

The following file should then be put in `/etc/cron.d/nginx-crl`

```
# This cron runs every 5 minutes and execute a script replacing the CRL file for NGINX
*/5 * * * * root /opt/stream/sbin/crl-fetching
```

Stacktraces management

Stacktraces in the functional logs can give a lot of information about the technical architecture of the application. To disable their logging, set the parameter `stream.event.disable-stacktrace` to **true** following the steps in the **Overridable configuration parameters** section of the administration guide.



Stacktraces are still available in the technical logs.

1.3. Installing on Kubernetes

1.3.1. Installation

Concepts overview

In Kubernetes, applications are deployed onto **Pods**, which represents a running version of a containerized application. Pods are grouped by **Deployments**, which represent a set of Pods running the same application. For instance, should you need to run Stream in high availability mode, your deployment will contain 3 pods or more. Applications running in Pods are made accessible by a **Service**, which grants a set of Pods an IP address (which can either be internal to the cluster or accessible on the public Internet through a Load Balancer).

The recommended way of installing on Stream is through the Stream's Helm Chart. Helm is a package manager for Kubernetes that will generate Kubernetes resources necessary to deploy Stream onto your cluster. The official Helm Chart will generate a deployment of one or more Pods running Stream on your cluster.

Setting up Helm repository

Now that the application secrets are configured, add the **EverTrust Helm repository** to your machine:

```
$ helm repo add evertrust https://repo.evertrust.io/repository/charts
```

Verify that you have access to the Chart :

```
$ helm search repo evertrust/stream
NAME                    CHART VERSION   APP VERSION DESCRIPTION
evertrust/stream        0.2.0           2.0.0       EverTrust Stream Helm chart
```

Configuring the namespace

For isolation purposes, we strongly recommend that you create a dedicated namespace for **Stream**:

```
$ kubectl create namespace stream
```

The namespace should be empty. In order to run Stream, you'll need to create two secrets in that namespace:

- A data secret containing your Stream license file and keyset.
- An image pull secret, allowing Kubernetes to authenticate to the EverTrust's container

repository

Creating the application secrets

Licence

You should have a license file for your Stream installation, most probably named `stream.lic`. To convert this file to a Kubernetes secret, run:

```
$ kubectl create secret generic stream-license \
  --from-file=license="<path to your license file>" \
  --namespace stream
```

Keyset

Stream requires a keyset in order to work. Follow the steps in the Tinkey page to setup Tinkey, create your Keyset, and make it available to Stream.

Creating the image pull secret

Next, you should configure Kubernetes to authenticate to the EVERTRUST repository using your credentials. They are necessary to pull the Stream docker image, you should have received them upon purchase. Get your username and password and create the secret:

```
$ kubectl create secret docker-registry evertrust-registry \
  --docker-server=registry.evertrust.io \
  --docker-username="<your username>" \
  --docker-password="<your password>" \
  --namespace stream
```

Configuring the chart

You'll next need to override the defaults `values.yaml` file of the Helm Chart to reference the secrets that we've created. We'll provide a minimal configuration for demonstration purposes, but please do follow our production setup guide before deploying for production.

Create a `override-values.yaml` file somewhere and paste this into the file:

```
image:
  pullSecrets:
    - evertrust-registry

license:
  secretName: stream-data
  secretKey: license

keyset:
```

```
  secretName: stream-data
  secretKey: keyset
```

To finish Stream's installation, simply run the following command:

```
$ helm install stream evertrust/stream -f override-values.yaml -n stream
```

Please allow a few minutes for the Stream instance to boot up. You are now ready to go on with the first login. This instance will allow you to test out if Stream is working correctly on your cluster. However, this installation is not production-ready. Follow our production checklist to make sure your instance is fit to run in your production environment.

1.3.2. First login

Fetching the default administrator password

Allow a few seconds for your Stream instance to boot up. You can then fetch the administrator password that has been generated for your instance using the command :

```
$ kubectl exec $(kubectl get pods -n stream -l "app.kubernetes.io/name=stream" --sort
-by={.status.podIP} -o jsonpath="{.items[0].metadata.name}") -n <namespace> -- /bin/sh
-c "cat /stream/adminPassword"
```



The default administrator credentials are:

- Login: `administrator`
- Password: the one you got from the command above

Manually creating the initial user

In case the automatic bootstrap process was disabled, you may need to manually create an administrator user. Launch a MongoDB shell to access your database and run the following command to create the initial administrator:

```
db.security_accounts.insertOne({"identifier":"administrator","secret":"$6$96ZV/UmX1oMP
UVA3$U5MejjbJ9S3jhqq1TDqhZMwVOcDX5BAWY3DL2nsxUHlpHj0LOfPuswy4nWjkMLify4FvKGKhEfADzljy7
FGc8.","permissions":[{"value":"configuration:*"},{"value":"lifecycle:*"}],"roles":[],
"type":"local"})
```

Set a default password

As mentioned above, when Stream first boots in an empty database, it generates a random password for the default `administrator` account. If you wish to deterministically set the password for this account, you can do so by setting the `initialAdminPassword` key in the Helm Chart's values

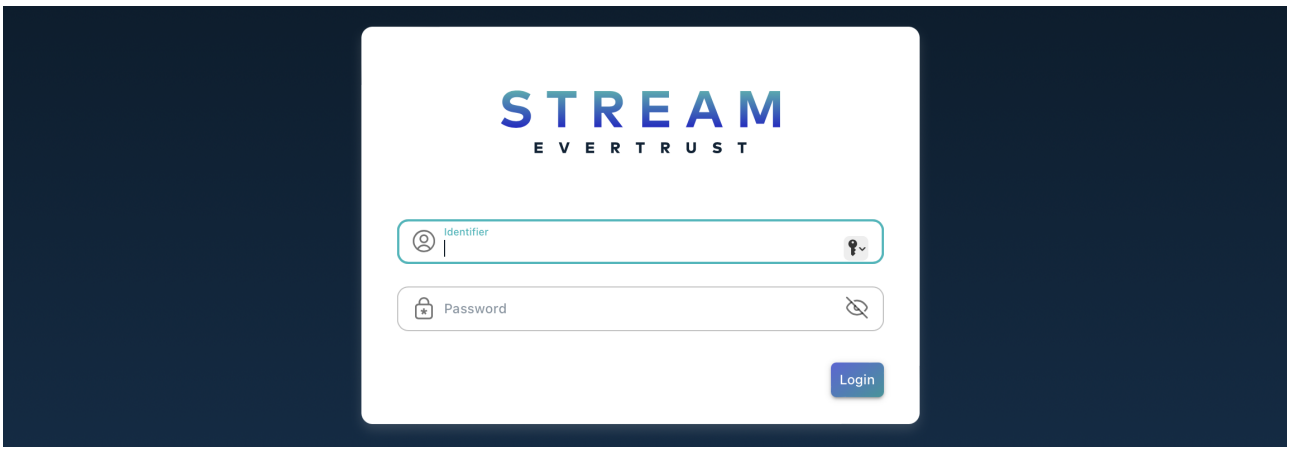
override, or set the `HRZ_ADMIN_PWD_HASH` environment variable.

Stream expects a SHA-crypt hash, which can be generated using our toolbox image:

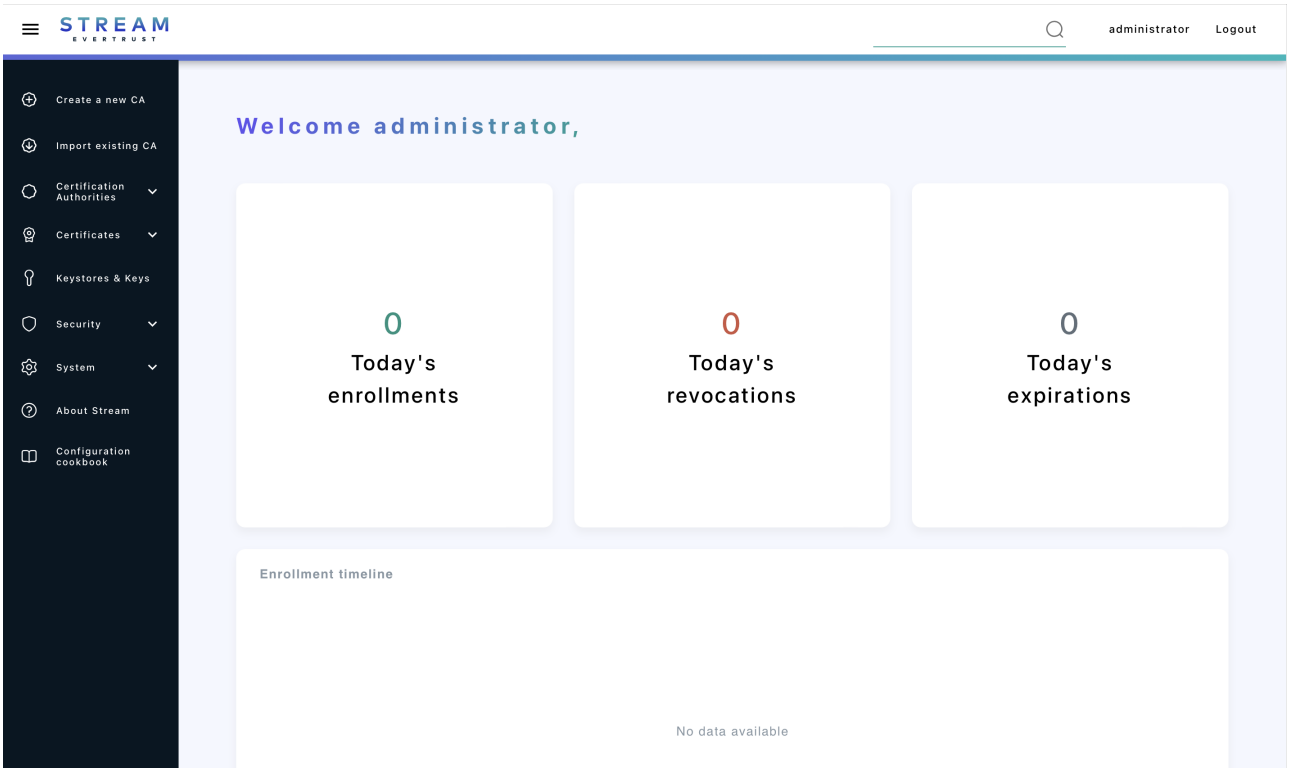
```
docker run --rm -it quay.io/evertrust/toolbox:latest generate-password sample_password
```

Accessing the Stream Web Interface

1. Launch a web browser;
2. Browse to `https://[IP or DNS Name of the Stream component]/ui#`:



3. Specify the default administration credentials and hit the '**Login**' button:





It is **highly recommended** to delete the `adminPassword` file from your machine once you saved it somewhere safe.

1.3.3. Production checklist

Even though the Helm Chart makes installing Stream a breeze, you'll still have to set up a few things to make Stream resilient enough to operate in a production environment.

Operating the database

All persistent data used by Stream is stored in the underlying MongoDB database. Therefore, the database should be operated securely and backed up regularly.

When installing the chart, you face multiple options regarding your database:

- By default, a temporary MongoDB instance will be spawned in your cluster, using the `mongo` image. No additional configuration is required but it is not production ready out of the box.
- If you want to use an existing MongoDB instance, provide the `externalDatabase.uri` value. The URI should be treated as a secret as it must include credentials:

```
externalDatabase:
  secretName: <secret name>
  secretKey: <secret key>
```

The chart doesn't manage the database. You are still in charge of making sure that the database is correctly backed up. You could either back up manually using `mongodump` or use a managed service such as MongoDB Atlas, which will take care of the backups for you.

Managing secrets

Storing secrets is a crucial part of your Stream installation. The keyset is the most import of them, being a master key used to encrypt and decrypt data before they enter the database. Alongside with other application secrets like your MongoDB URI (containing your credentials or certificate). We recommend that you create Kubernetes secrets beforehand or inject them directly into the pod.

Values that should be treated as secrets in this chart are:

Name	Description	Impact on loss
`keyset`	Master key used to encrypt sensitive data in database.	Highest impact: database would be unusable
`events.secret`	Secret used to sign and chain events.	Moderate impact: events integrity would be unverifiable
`externalDatabase.uri`	External database URI, containing a username and password.	Low impact: reset the MongoDB password
`appSecret`	Application secret use to encrypt session data.	Low impact: sessions would be reset

Name	Description	Impact on loss
`mailer.password`	SMTP server password	Low impact: reset the SMTP password

For each of these values, either :

- leave the field empty, so that a secret will be automatically generated.
- derive the secret value from an existing Kubernetes secret:

```
appSecret:
  secretName: <secret name>
  secretKey: <secret key>
```



Always store secrets in a safe place after they're generated. If you ever uninstall your Helm chart, the loss of the keyset will lead to the impossibility of recovering most of your data.

High availability

By default, the chart will configure a single-pod deployment. This deployment method is fine for testing but not ready for production as a single failure could take down the entire application. Instead, we recommend that you set up a Stream cluster using at least 3 pods.

In order to do that, configure an `horizontalAutoscaler` in your `override-values.yaml` file:

```
horizontalAutoscaler:
  enabled: true
  minReplicas: 3
  maxReplicas: 3
```



Use `nodeAffinity` to spread your Stream cluster Pods among multiple nodes in different availability zones to reduce the risk of Single Point of Failure.

Configuring ingresses

To create an ingress upon installation, simply set the following keys in your `override-values.yaml` file:

```
ingress:
  enabled: true
  hostname: stream.lab
  tls: true
```

1.3.4. Advanced usage

Some edge use-cases might not have been included in the previous installation documentation, for clarity purposes. You may find some of them below.

Running behind a container registry proxy

If your installation environment requires you to whitelist images that can be pulled by the Kubernetes cluster, you must whitelist the `registry.evertrust.io/stream` and `registry.evertrust.io/stream-upgrade` images. It is then possible to override the images being pulled by setting the `global.imageRegistry` key in your `values.yaml` file to point to your private registry:

```
global:
  imageRegistry: <YOUR-PRIVATE-REGISTRY>
```

Leases

To ensure clustering issues get resolved as fast as possible, Stream can use Kubernetes leases. We strongly recommend that you use this safety mechanism. However, the feature can be disabled by setting the `leases.enabled` key to `false`.

1.3.5. Chart reference

The Helm chart aims to offer easy config settings through values for most used Kubernetes or Stream features. If something you need is not yet covered, you can also use:

- `extraConfig` to directly edit Stream config;
- `extraObjects` to add related Kubernetes resources.

The chart source is also freely available to investigate unwanted behaviors.

kubeVersion

Force target Kubernetes version (using Helm capabilities if not set)

```
kubeVersion: ""
```

nameOverride

String to partially override stream.fullname

```
nameOverride: ""
```

fullnameOverride

String to fully override stream.fullname

```
fullnameOverride: ""
```

imageRegistry

String to override the image registry for all containers

```
imageRegistry: ""
```

commonLabels

Labels to add to all deployed objects

```
commonLabels: {}
```

commonAnnotations

Annotations to add to all deployed objects

```
commonAnnotations: {}
```

image

By default, we fetch the Stream image from the Evertrust registry. If the tag is null or unset, the default value will be set the to the chart appVersion. As the official Evertrust registry is not in open-access, you should specify an image pull secret that has access to Stream images.

Kubernetes Reference

registry

Stream image registry

```
image.registry: "registry.evertrust.io"
```

repository

Stream image repository

```
image.repository: "stream"
```

tag

Stream image tag (immutable tags are recommended)

```
image.tag: "2.1.8"
```

flavor

Stream image flavor (for HSM compatible images)

```
image.flavor: ""
```

pullPolicy

Stream image pull policy

```
image.pullPolicy: "IfNotPresent"
```

pullSecrets

Stream image pull secrets

```
image.pullSecrets: []
```

updateStrategy



Set it to `Recreate` if you use a PV that cannot be mounted on multiple pods

Kubernetes Reference

Example

```
updateStrategy:
 type: RollingUpdate
 rollingUpdate:
   maxSurge: 25%
   maxUnavailable: 25%
```

type

Stream deployment strategy type

```
updateStrategy.type: "Recreate"
```

deploymentAnnotations

Annotations to add to the deployment object

```
deploymentAnnotations: {}
```

deploymentLabels

Annotations to add to the deployment object

```
deploymentLabels: {}
```

priorityClassName

Stream pod priority class name

```
priorityClassName: ""
```

hostAliases

Stream pod host aliases

Kubernetes Reference

```
hostAliases: []
```

extraVolumes

Optionally specify extra list of additional volumes for Stream pods

```
extraVolumes: []
```

Example

```
extraVolumes:
  - name: extra-volume-name
    configMap:
      name: example-configmap
```

extraVolumeMounts

Optionally specify extra list of additional volumeMounts for Stream container(s)

```
extraVolumeMounts: []
```

Example

```
extraVolumeMounts:
  - name: extra-volume-name
    mountPath: /mnt/extra-volume
```

sidecars

Add additional sidecar containers to the Stream pod

```
sidecars: []
```

Example

```
sidecars:
  - name: your-image-name
    image: your-image
    imagePullPolicy: Always
    ports:
      - name: portname
        containerPort: 1234
```

initContainers

Add additional init containers to the Stream pod

```
initContainers: []
```

lifecycleHooks

Add lifecycle hooks to the Stream deployment

```
lifecycleHooks: {}
```

podLabels

Extra labels for Stream pods

Kubernetes Reference

```
podLabels: {}
```

podAnnotations

Annotations for Stream pods

Kubernetes Reference

```
podAnnotations: {}
```

podAffinityPreset

Pod affinity preset. Ignored if `affinity` is set. Allowed values: `soft` or `hard`

Kubernetes Reference

```
podAffinityPreset: ""
```

podAntiAffinityPreset

Pod anti-affinity preset. Ignored if `affinity` is set. Allowed values: `soft` or `hard`

Kubernetes Reference

```
podAntiAffinityPreset: "soft"
```

nodeAffinityPreset

Node affinity preset

Kubernetes Reference

type

Node affinity preset type. Ignored if `affinity` is set. Allowed values: `soft` or `hard`

```
nodeAffinityPreset.type: ""
```

key

Node label key to match. Ignored if `affinity` is set

```
nodeAffinityPreset.key: ""
```

values

Node label values to match. Ignored if `affinity` is set

```
nodeAffinityPreset.values: []
```

Example

```
values:
  - e2e-az1
  - e2e-az2
```

revisionHistoryLimit

Number of controller revisions to keep

```
revisionHistoryLimit: 3
```

replicas

Replica count when no autoscaler is configured

```
replicas: 1
```

affinity



podAffinityPreset, podAntiAffinityPreset, and nodeAffinityPreset will be ignored when it's set

Kubernetes Reference

```
affinity: {}
```

nodeSelector

Node labels for pod assignment

Kubernetes Reference

```
nodeSelector: {}
```

tolerations

Tolerations for pod assignment

Kubernetes Reference

```
tolerations: []
```

topologySpreadConstraints

Spread Constraints for pod assignment

Kubernetes Reference

```
topologySpreadConstraints: []
```

Example

```
topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: node
    whenUnsatisfiable: DoNotSchedule
```

resources

Stream containers' resource requests and limits

The JVM will automatically adapt the memory allocation pool to the container allocated resources.

Kubernetes Reference

limits

The resources limits for the Stream container

```
resources.limits: {}
```

requests

The requested resources for the Stream container

```
resources.requests:
```

```
  memory: 512Mi
  cpu: 300m
```

podSecurityContext

Configure Pods Security Context

Kubernetes Reference

enabled

Enabled Stream pods' Security Context

```
podSecurityContext.enabled: true
```

fsGroup

Set Stream pod’s Security Context fsGroup

```
podSecurityContext.fsGroup: 1001
```

containerSecurityContext

Configure Container Security Context (only main container)

Kubernetes Reference

enabled

Enabled Stream containers' Security Context

```
containerSecurityContext.enabled: true
```

runAsUser

Set Stream container’s Security Context runAsUser

```
containerSecurityContext.runAsUser: 1001
```

runAsNonRoot

Set Stream container’s Security Context runAsNonRoot

```
containerSecurityContext.runAsNonRoot: true
```

livenessProbe

Configure extra options for Stream containers probes

Kubernetes Reference

enabled

Enable livenessProbe

```
livenessProbe.enabled: true
```

initialDelaySeconds

Initial delay seconds for livenessProbe

```
livenessProbe.initialDelaySeconds: 0
```

periodSeconds

Period seconds for livenessProbe

```
livenessProbe.periodSeconds: 10
```

timeoutSeconds

Timeout seconds for livenessProbe

```
livenessProbe.timeoutSeconds: 5
```

successThreshold

Success threshold for livenessProbe

```
livenessProbe.successThreshold: 1
```

failureThreshold

Failure threshold for livenessProbe

```
livenessProbe.failureThreshold: 3
```

startupProbe

A startup probe allows us to define a shorter period to improve Stream time-to-liveliness time while preserving the Stream pod from a restart loop when it is slow to start.

Kubernetes Reference

enabled

Enable startupProbe. Since Stream is slow to start, this is highly recommended.

```
startupProbe.enabled: true
```

failureThreshold

Failure threshold for startupProbe

```
startupProbe.failureThreshold: 60
```

periodSeconds

Period seconds for startupProbe

```
startupProbe.periodSeconds: 3
```

horizontalAutoscaler

Stream Autoscaling configuration

Kubernetes Reference

enabled

Enable Horizontal POD autoscaling for Stream

```
horizontalAutoscaler.enabled: false
```

minReplicas

Minimum number of Stream replicas

```
horizontalAutoscaler.minReplicas: 1
```

maxReplicas

Maximum number of Stream replicas

```
horizontalAutoscaler.maxReplicas: 3
```

targetCPU

Target CPU utilization percentage

```
horizontalAutoscaler.targetCPU: 50
```

targetMemory

Target Memory utilization percentage

```
horizontalAutoscaler.targetMemory: 50
```

disruptionBudget

Pod Disruption Budget configuration

Kubernetes Reference

enabled

Created a PodDisruptionBudget

```
disruptionBudget.enabled: false
```

minAvailable

Min number of pods that must still be available after the eviction

```
disruptionBudget.minAvailable: 1
```

maxUnavailable

Max number of pods that can be unavailable after the eviction

```
disruptionBudget.maxUnavailable: 0
```

environment

Configure environment variable injections into Stream's pods. This is the way you should inject secrets into the app if you wish to use the Kubernetes secrets implementation.

Kubernetes Reference

```
environment: []
```

dnsConfig

Pod's DNS Configuration

This value is useful if you need to resolve your custom domain for ACME challenges

Kubernetes Reference

```
dnsConfig: {}
```

Example

```
nameservers:
  - 1.2.3.4
searches:
  - ns1.svc.cluster-domain.example
  - my.dns.search.suffix
options:
  - name: ndots
    value: "2"
```

dnsPolicy

Pod's DNS Policy

Kubernetes Reference

```
dnsPolicy: "ClusterFirst"
```

service

Service configuration

type

Kubernetes service type

```
service.type: "ClusterIP"
```

clusterIP

Stream service clusterIP IP

```
service.clusterIP: ""
```

Example

```
clusterIP: None
```

loadBalancerIP

Load balancer IP for the Stream Service (optional, cloud specific)

Kubernetes Reference

```
service.loadBalancerIP: ""
```

loadBalancerSourceRanges

Address that are allowed when service is LoadBalancer

Kubernetes Reference

```
service.loadBalancerSourceRanges: []
```

Example

```
loadBalancerSourceRanges:
  - 10.10.10.0/24
```

externalTrafficPolicy

Enable client source IP preservation

Kubernetes Reference

```
service.externalTrafficPolicy: "Cluster"
```

extraPorts

Extra port to expose on Stream service

```
service.extraPorts: []
```

annotations

Annotations for Stream service

```
service.annotations: {}
```

ingress

Ingress configuration

Kubernetes Reference

enabled

Set to true to enable ingress record generation

```
ingress.enabled: false
```

ingressClassName

IngressClass that will be used to implement the Ingress (Kubernetes 1.18+)

```
ingress.ingressClassName: ""
```

hostname

Default host for the ingress resource

```
ingress.hostname: ""
```

path

Default path for the ingress record NOTE: You may need to set this to '/*' in order to use this with ALB ingress controllers

```
ingress.path: "/"
```

pathType

Ingress path type

```
ingress.pathType: "Prefix"
```

annotations

Additional annotations for the Ingress resource. To enable certificate autogeneration, place here your cert-manager annotations.

```
ingress.annotations: {}
```

Example

```
annotations:
  cert-manager.io/cluster-issuer: cluster-issuer-name
```

tls

Enable TLS configuration for the hostname defined at ingress.hostname TLS certificates will be retrieved from a TLS secret with name: {{- printf "%s-tls" .Values.ingress.hostname }} You can use the ingress.secrets parameter to create this TLS secret, relay on cert-manager to create it, or let the chart create self-signed certificates for you

```
ingress.tls: false
```

extraHosts

Additional hostnames to be covered with this ingress record Most likely the hostname above will be enough, but in the event more hosts are needed, this is an array

```
ingress.extraHosts: []
```

Example

```
extraHosts:
- name: stream.local
  path: /
```

extraPaths

An array with additional arbitrary paths that may need to be added to the ingress under the main host

```
ingress.extraPaths: []
```

Example

```
extraPaths:
- path: /*
  backend:
    serviceName: ssl-redirect
    servicePort: use-annotation
```

extraTls

The tls configuration for additional hostnames to be covered with this ingress record.

Kubernetes Reference

```
ingress.extraTls: []
```

Example

```
extraTls:
- hosts:
    - stream.local
  secretName: stream.local-tls
```

extraRules

Additional rules to be covered with this ingress record

Kubernetes Reference

```
ingress.extraRules: []
```

Example

```
extraRules:
- host: stream.local
    http:
      path: /
      backend:
        service:
          name: stream
          port:
            name: http
```

appSecret

Configure the Play secret for the Stream instance. As this is used for cryptographic purposes, it should be fetched from the environment.

https://www.playframework.com/documentation/2.8.x/ApplicationSecret

```
appSecret:
  secretName: ''
  secretKey: ''
```

Example

```
appSecret:
  valueFrom:
    secretKeyRef:
      name: stream-secret
      key: appSecret
```

license

A valid Stream license is required for the software to run. You should store it (base64-encoded) in a Kubernetes secret and specify the secret details here.

README.md[README.md]

secretName

Existing secret name where the Stream license is stored

```
license.secretName: ""
```

secretKey

Existing secret key where the Stream license is stored

```
license.secretKey: ""
```

initialAdminHashPassword

Set up initial admin user.

enabled

Whether to enable the initial admin user

```
initialAdminHashPassword.enabled: false
```

secretName

Existing secret name where the initial admin password is stored

```
initialAdminHashPassword.secretName: ""
```

secretKey

Existing secret key where the initial admin password is stored

```
initialAdminHashPassword.secretKey: ""
```

allowedHosts

Additional allowed hosts that are whitelisted to access the Stream UI. Configured ingresses will automatically be added to the list, this should only be used when port forwarding or when an ingress is created manually.

```
allowedHosts:
  - localhost:9000
  - localhost:9443
```

Example

```
allowedHosts:
  - localhost:9000
  - demo.example.org
```

trustedProxies

Depending on your Kubernetes environment, Ingress IPs may be unpredictable. In that case, you should trust whitelist every IP in your local addressing space.

```
trustedProxies:
  - 0.0.0.0/0
  - '::/0'
```

Example

```
trustedProxies:
  - 0.0.0.0/0
```

```
  - ::/0
```

events

Configuration for Stream events

chainsign

Whether Stream events should be signed and chained using the event seal secret.

```
events.chainsign: true
```

ttl

Duration during which events are kept in database.

```
events.ttl: "90 days"
```

logFormat

Format in which logs will be outputted. Can be set either to "console" or "json" for structured logging.

```
logFormat: "console"
```

serviceAccount

Stream pods ServiceAccount

Kubernetes Reference

create

Enable the creation of a ServiceAccount for Stream pods

```
serviceAccount.create: true
```

name

Name of the created ServiceAccount

If not set and create is true, a name is generated using the stream.fullname template

```
serviceAccount.name: ""
```

annotations

Annotations for Stream Service Account

```
serviceAccount.annotations: {}
```

automountServiceAccountToken

Automount service account token for the server service account

```
serviceAccount.automountServiceAccountToken: true
```

clientCertificateHeader

Indicates to Stream in which header the client certificate will be passed by the Ingress controller.

```
clientCertificateHeader: ""
```

podsDirectConnect

Whether Stream pods should connect to each other directly via IP, or through a DNS record generated by a Kubernetes DNS server. Useful if the kube-dns server is configured with "pods disabled" or if you use GKE Cloud DNS NOTE: This is not support by Istio

```
podsDirectConnect: false
```

extraConfig

Additional configuration for Stream. Injecting arbitrary config could result in unexpected behavior. Proceed with caution.

https://docs.evertrust.fr/stream/admin-guide/-/parameters

```
extraConfig: ""
```

Example

```
extraConfig: |
  stream {
    notification.mail.attachment.extension.der = "der"
```

```
  }
```

externalDatabase

Configuration for a Stream external database Refer to the Stream installation guide to configure the installation correctly

secretName

Existing secret name where the external MongoDB URI is stored

```
externalDatabase.secretName: ""
```

secretKey

Existing secret key where the external MongoDB URI is stored

```
externalDatabase.secretKey: ""
```

mailer

Configuration for the Stream mailer. You should configure this if you want your Stream instance to be able to send emails. You should fetch credentials from the environment if they are required.

host

SMTP host

```
mailer.host: ""
```

port

SMTP host port

```
mailer.port: ""
```

tls

Enable TLS for this SMTP host

```
mailer.tls: ""
```

ssl

Enable SSL for this SMTP host

```
mailer.ssl: ""
```

user

Authentication username for this SMTP host

```
mailer.user: ""
```

password

The password field can be a reference to a secret.

```
mailer.password: {}
```

Example

```
password:
  valueFrom:
    secretKeyRef:
      name: stream-secret
      key: mailerPassword
```

extraObjects

Create dynamic manifests via values

```
extraObjects: []
```

Example

```
- apiVersion: "kubernetes-client.io/v1"
  kind: ExternalSecret
  metadata:
    name: stream-secrets
  spec:
    backendType: gcpSecretsManager
    data:
      - key: stream-secret-key
        name: stream-secret-name
```

metrics

Enable Prometheus metrics

enabled

Whether to enable Prometheus metrics

```
metrics.enabled: false
```

port

Prometheus metrics port

```
metrics.port: 9095
```

1.4. Tinkey

This part describes how to generate a Tink Keyset and configure it in Stream

Installation

RHEL



Tinkey requires java 17 or 21 installed

Installation from the EverTrust repository

Create a `/etc/yum.repos.d/tinkey.repo` file containing the EverTrust repository info:

```
[tinkey]
enabled=1
name=Tinkey Repository
baseurl=https://repo.evertrust.io/repository/tinkey-rpm/
gpgcheck=1
gpgkey=https://evertrust.io/.well-known/rpm/gpg.pub
username=<username>
password=<password>
```

Replace `<username>` and `<password>` with the credentials you were provided.

You can then run the following to install the latest Tinkey version:

```
# yum install tinkey
```

To prevent unattended upgrades when running yum update, you should pin the Tinkey version by adding

```
exclude=tinkey
```

at the end of the `/etc/yum.repos.d/tinkey.repo` file after installing Tinkey.

Installing from RPM

Download the latest RPM for Tinkey on the Official EVERTRUST repository.

Upload the file '***tinkey-<latest>.noarch.rpm***' to the server;

Access the server with an account with administrative privileges;

Install the Tinkey package with the following command:

```
# yum localinstall /root/tinkey-<latest>.noarch.rpm
```

If you wish to verify the signature of the RPM package, the EVERTRUST key can be added to your trusted keys using the following command:

```
# rpm --import https://evertrust.io/.well-known/rpm/gpg.pub
```

The signature can then be verified using the following command:

```
# rpm -K /root/tinkey-<latest>.noarch.rpm
```

Debian



Tinkey requires java 17 or 21 installed

Installation from the EverTrust repository

If you haven't already, to add the EVERTRUST repository to your APT repositories, run the following commands:

1. Install the required tools (`gpg`)

```
# sudo apt install gnupg
```

2. Download and install the EVERTRUST GPG key

```
# curl https://evertrust.io/.well-known/apt/gpg.pub | sudo gpg -o
/usr/share/keyrings/evertrust.gpg --dearmor
```

3. Add the repository

```
# echo "deb [ arch=all signed-by=/usr/share/keyrings/evertrust.gpg ]
https://repo.evertrust.io/repository/apt all main" | sudo tee
/etc/apt/sources.list.d/evertrust.list
```

Once the repository has been added, authentication to it must be provided. To do so, edit the `/etc/apt/auth.conf` file and add the following lines:

```
machine repo.evertrust.io
login <your EVERTRUST login>
password <your EVERTRUST password>
```

Once the repository has been added, run the following command to update the APT repository list.

```
# sudo apt update
```

You can then run the following command to install the latest Tinkey version:

```
# sudo apt install tinkey
```

Installing from DEB

Download the latest DEB for Tinkey on the Official EVERTRUST repository.

Upload the file '***tinkey-<latest>_all.deb***' to the server;

Access the server with an account with administrative privileges;

Install the Tinkey package with the following command:

```
# apt install /root/tinkey-<latest>_all.deb
```

Docker

The EVERTRUST Tinkey utility is available at:

```
registry.evertrust.io/tinkey
```

Usage

PlainText keyset

In this mode, the keyset directly contains an AES key without additional encryption. When Stream starts, the keyset is loaded into memory and used for all encryption and decryption operations.

RHEL

```
/opt/evertrust/tinkey/bin/tinkey create-keyset --key-template AES256_GCM
--out=stream.keyset
```

Debian

```
/opt/evertrust/tinkey/bin/tinkey create-keyset --key-template AES256_GCM
--out=stream.keyset
```

Docker

```
docker run registry.evertrust.io/tinkey create-keyset --key-template AES256_GCM
--out=stream.keyset
```



This is the equivalent to previous Stream **SSV** encryption level.

PKCS#11 keyset

PKCS#11 templates enable integration with Hardware Security Modules (HSM) for enhanced security. Stream supports three different PKCS#11 modes depending on your security requirements.

The following parameters are commonly used when generating the key-uri:

Parameter	Description
object	The label of the symmetric key. **The key must already exist on the HSM**.
type	The type of key (typically "secret-key")
slot-id	The HSM slot identifier to use
module-path	The path to the .so library required for HSM interaction
pin-value	The PIN required to authenticate with the HSM

Wrapped mode

In this mode, a software-based AES key is encrypted (wrapped) with a master key stored in the HSM. The wrapped key is stored in the keyset file, while the master key remains securely in the HSM. When Stream starts, the keyset is decrypted using the HSM's master key, then loaded into memory and used for all subsequent encryption and decryption operations. This approach

balances security with performance by minimizing HSM communication while keeping the master key protected in hardware.

To create a wrapped keyset using the GCM algorithm:

RHEL

```
/opt/evertrust/tinkey/bin/tinkey create-keyset --key-template AES128_GCM --out
stream.keyset --master-key-uri "pkcs11://object=AES1;type=secret-key;slot-id=-
1?module-path=/usr/lib64/pkcs11/libsofthsm2.so&pin-value=1234"
```

Debian

```
/opt/evertrust/tinkey/bin/tinkey create-keyset --key-template AES128_GCM --out
stream.keyset --master-key-uri "pkcs11://object=AES1;type=secret-key;slot-id=-
1?module-path=/usr/lib64/pkcs11/libsofthsm2.so&pin-value=1234"
```

Docker

```
docker run registry.evertrust.io/tinkey create-keyset --key-template AES128_GCM
--out stream.keyset --master-key-uri "pkcs11://object=AES1;type=secret-
key;slot-id=-1?module-path=/usr/lib64/pkcs11/libsofthsm2.so&pin-value=1234"
```

To use the CBC algorithm instead, replace `pkcs11://` with `pkcs11-aes-cbc://` in the master-key-uri:

RHEL

```
/opt/evertrust/tinkey/bin/tinkey create-keyset --key-template AES128_GCM --out
stream.keyset --master-key-uri "pkcs11-aes-cbc://object=AES1;type=secret-
key;slot-id=-1?module-path=/usr/lib64/pkcs11/libsofthsm2.so&pin-value=1234"
```

Debian

```
/opt/evertrust/tinkey/bin/tinkey create-keyset --key-template AES128_GCM --out
stream.keyset --master-key-uri "pkcs11-aes-cbc://object=AES1;type=secret-
key;slot-id=-1?module-path=/usr/lib64/pkcs11/libsofthsm2.so&pin-value=1234"
```

Docker

```
docker run registry.evertrust.io/tinkey create-keyset --key-template AES128_GCM
--out stream.keyset --master-key-uri "pkcs11-aes-cbc://object=AES1;type=secret-
key;slot-id=-1?module-path=/usr/lib64/pkcs11/libsofthsm2.so&pin-value=1234"
```



When using wrapped mode, Stream needs the master-key-uri to decrypt the keyset

> at startup. Please follow the configuration steps below.

Hardware protected mode

In this mode, the encryption key is stored directly in the HSM. All encryption and decryption operations are performed by the HSM itself, ensuring the key never leaves the hardware security boundary. This provides a high level of security but requires HSM communication for every cryptographic operation.

To create the keyset using the GCM algorithm:

RHEL

```
/opt/evertrust/tinkey/bin/tinkey create-keyset --key-template PKCS11_AES_GCM
--out keyset --key-uri pkcs11://object=AES1;type=secret-key;slot-id=-1?module-
path=/usr/lib64/pkcs11/libsofthsm2.so&pin-value=1234
```

Debian

```
/opt/evertrust/tinkey/bin/tinkey create-keyset --key-template PKCS11_AES_GCM
--out keyset --key-uri pkcs11://object=AES1;type=secret-key;slot-id=-1?module-
path=/usr/lib64/pkcs11/libsofthsm2.so&pin-value=1234
```

Docker

```
docker run registry.evertrust.io/tinkey create-keyset --key-template
PKCS11_AES_GCM --out keyset --key-uri pkcs11://object=AES1;type=secret-
key;slot-id=-1?module-path=/usr/lib64/pkcs11/libsofthsm2.so&pin-value=1234
```

To use the CBC algorithm instead, change the `--key-template` parameter to `PKCS11_AES_CBC`:

RHEL

```
/opt/evertrust/tinkey/bin/tinkey create-keyset --key-template PKCS11_AES_CBC
--out keyset --key-uri pkcs11://object=AES1;type=secret-key;slot-id=-1?module-
path=/usr/lib64/pkcs11/libsofthsm2.so&pin-value=1234
```

Debian

```
/opt/evertrust/tinkey/bin/tinkey create-keyset --key-template PKCS11_AES_CBC
--out keyset --key-uri pkcs11://object=AES1;type=secret-key;slot-id=-1?module-
path=/usr/lib64/pkcs11/libsofthsm2.so&pin-value=1234
```

Docker

```
docker run registry.evertrust.io/tinkey create-keyset --key-template
PKCS11_AES_CBC --out keyset --key-uri pkcs11://object=AES1;type=secret-
key;slot-id=-1?module-path=/usr/lib64/pkcs11/libsofthsm2.so&pin-value=1234
```

Derived hardware protected mode

In this mode, a master key stored in the HSM is used to derive encryption keys on demand. Each time Stream needs to encrypt or decrypt data, it derives a unique key from the master key in the HSM using a random seed. This seed is then stored alongside the encrypted data. This provides the highest level of security but requires HSM communication for every cryptographic operation.

To create the keyset using the GCM algorithm:

RHEL

```
/opt/evertrust/tinkey/bin/tinkey create-keyset --key-template
PKCS11_AES_GCM_DERIVED --out keyset --key-uri pkcs11://object=AES1;type=secret-
key;slot-id=-1?module-path=/usr/lib64/pkcs11/libsofthsm2.so&pin-value=1234
```

Debian

```
/opt/evertrust/tinkey/bin/tinkey create-keyset --key-template
PKCS11_AES_GCM_DERIVED --out keyset --key-uri pkcs11://object=AES1;type=secret-
key;slot-id=-1?module-path=/usr/lib64/pkcs11/libsofthsm2.so&pin-value=1234
```

Docker

```
docker run registry.evertrust.io/tinkey create-keyset --key-template
PKCS11_AES_GCM_DERIVED --out keyset --key-uri pkcs11://object=AES1;type=secret-
key;slot-id=-1?module-path=/usr/lib64/pkcs11/libsofthsm2.so&pin-value=1234
```

To use the CBC algorithm instead, change the `--key-template` parameter to `PKCS11_AES_CBC_DERIVED`:

RHEL

```
/opt/evertrust/tinkey/bin/tinkey create-keyset --key-template
PKCS11_AES_CBC_DERIVED --out keyset --key-uri pkcs11://object=AES1;type=secret-
key;slot-id=-1?module-path=/usr/lib64/pkcs11/libsofthsm2.so&pin-value=1234
```

Debian

```
/opt/evertrust/tinkey/bin/tinkey create-keyset --key-template
PKCS11_AES_CBC_DERIVED --out keyset --key-uri pkcs11://object=AES1;type=secret-
```

```
key;slot-id=-1?module-path=/usr/lib64/pkcs11/libsofthsm2.so&pin-value=1234
```

Docker

```
docker run registry.evertrust.io/tinkey create-keyset --key-template
PKCS11_AES_CBC_DERIVED --out keyset --key-uri pkcs11://object=AES1;type=secret-
key;slot-id=-1?module-path=/usr/lib64/pkcs11/libsofthsm2.so&pin-value=1234
```



The `PKCS11_AES_CBC_DERIVED` parameter is the equivalent to previous Stream **SHV** encryption level.

AWS KMS keyset

AWS KMS (Key Management Service) enables secure key management using Amazon's cloud infrastructure. In this mode, a software-based AES key is encrypted (wrapped) with a master key stored in AWS KMS. When Stream starts, the keyset is decrypted using the KMS master key, then loaded into memory and used for all subsequent encryption and decryption operations.

The following parameters are required when generating the AWS KMS key-uri:

Parameter	Description
key-uri	The AWS KMS key ARN (Amazon Resource Name) or alias
credentials	Path to AWS credentials file (optional if using IAM roles or environment variables)

To create an AWS KMS wrapped keyset:

RHEL

```
/opt/evertrust/tinkey/bin/tinkey create-keyset --key-template AES256_GCM --out
stream.keyset --master-key-uri "aws-kms://arn:aws:kms:us-east-
1:123456789012:key/12345678-1234-1234-1234-123456789012"
```

Debian

```
/opt/evertrust/tinkey/bin/tinkey create-keyset --key-template AES256_GCM --out
stream.keyset --master-key-uri "aws-kms://arn:aws:kms:us-east-
1:123456789012:key/12345678-1234-1234-1234-123456789012"
```

Docker

```
docker run registry.evertrust.io/tinkey create-keyset --key-template AES256_GCM
--out stream.keyset --master-key-uri "aws-kms://arn:aws:kms:us-east-
```

```
1:123456789012:key/12345678-1234-1234-1234-123456789012"
```

If you need to specify AWS credentials explicitly:

RHEL

```
/opt/evertrust/tinkey/bin/tinkey create-keyset --key-template AES256_GCM --out
stream.keyset --master-key-uri "aws-kms://arn:aws:kms:us-east-
1:123456789012:key/12345678-1234-1234-1234-123456789012" --credentials
/path/to/credentials.json
```

Debian

```
/opt/evertrust/tinkey/bin/tinkey create-keyset --key-template AES256_GCM --out
stream.keyset --master-key-uri "aws-kms://arn:aws:kms:us-east-
1:123456789012:key/12345678-1234-1234-1234-123456789012" --credentials
/path/to/credentials.json
```

Docker

```
docker run registry.evertrust.io/tinkey create-keyset --key-template AES256_GCM
--out stream.keyset --master-key-uri "aws-kms://arn:aws:kms:us-east-
1:123456789012:key/12345678-1234-1234-1234-123456789012" --credentials
/path/to/credentials.json
```



When using AWS KMS keyset, Stream needs the master-key-uri to decrypt the keyset at startup. Please follow the configuration steps below. Additionally, ensure your AWS credentials are properly configured (via service account, application default credentials, or credentials file).

GCP KMS keyset

GCP KMS (Google Cloud Key Management Service) provides cloud-based key management using Google Cloud Platform. In this mode, a software-based AES key is encrypted (wrapped) with a master key stored in GCP KMS. When Stream starts, the keyset is decrypted using the KMS master key, then loaded into memory and used for all subsequent encryption and decryption operations.

The following parameters are required when generating the GCP KMS key-uri:

Parameter	Description
key-uri	The GCP KMS key resource name in the format: `projects/PROJECT_ID/locations/LOCATION/keyRings/KEY_RING/cryptoKeys/KEY_NAME`

Parameter	Description
credentials	Path to GCP service account credentials JSON file (optional if using default application credentials)

To create a GCP KMS wrapped keyset:

RHEL

```
/opt/evertrust/tinkey/bin/tinkey create-keyset --key-template AES256_GCM --out
stream.keyset --master-key-uri "gcp-kms://projects/my-project/locations/us-
east1/keyRings/my-keyring/cryptoKeys/my-key"
```

Debian

```
/opt/evertrust/tinkey/bin/tinkey create-keyset --key-template AES256_GCM --out
stream.keyset --master-key-uri "gcp-kms://projects/my-project/locations/us-
east1/keyRings/my-keyring/cryptoKeys/my-key"
```

Docker

```
docker run registry.evertrust.io/tinkey create-keyset --key-template AES256_GCM
--out stream.keyset --master-key-uri "gcp-kms://projects/my-
project/locations/us-east1/keyRings/my-keyring/cryptoKeys/my-key"
```

If you need to specify GCP credentials explicitly:

RHEL

```
/opt/evertrust/tinkey/bin/tinkey create-keyset --key-template AES256_GCM --out
stream.keyset --master-key-uri "gcp-kms://projects/my-project/locations/us-
east1/keyRings/my-keyring/cryptoKeys/my-key" --credentials /path/to/service-
account.json
```

Debian

```
/opt/evertrust/tinkey/bin/tinkey create-keyset --key-template AES256_GCM --out
stream.keyset --master-key-uri "gcp-kms://projects/my-project/locations/us-
east1/keyRings/my-keyring/cryptoKeys/my-key" --credentials /path/to/service-
account.json
```

Docker

```
docker run registry.evertrust.io/tinkey create-keyset --key-template AES256_GCM
--out stream.keyset --master-key-uri "gcp-kms://projects/my-
project/locations/us-east1/keyRings/my-keyring/cryptoKeys/my-key" --credentials
```

```
/path/to/service-account.json
```



When using GCP KMS keyset, Stream needs the master-key-uri to decrypt the keyset at startup. Please follow the configuration steps below. Additionally, ensure your GCP credentials are properly configured (via service account, application default credentials, or credentials file).

Stream Configuration

Keyset

Once the keyset is created, it must be made available to Stream:

RPM

Move it to the expected location on the Stream server:

```
mv stream.keyset /opt/stream/etc/stream.keyset
```

Then set the appropriate ownership and permissions:

```
chown stream:stream /opt/stream/etc/stream.keyset
chmod 660 /opt/stream/etc/stream.keyset
```

Debian

Move it to the expected location on the Stream server:

```
mv stream.keyset /opt/stream/etc/stream.keyset
```

Then set the appropriate ownership and permissions:

```
chown stream:stream /opt/stream/etc/stream.keyset
chmod 660 /opt/stream/etc/stream.keyset
```

Kubernetes

Create a Kubernetes secret containing the keyset on the Stream namespace :

```
$ kubectl create secret generic stream-keyset \
  --from-file=keyset="<path to your keyset file>" \
  --namespace stream
```

Then reference the keyset in your `values.yaml`:

```
keyset:
  secretName: stream-keyset
  secretKey: keyset
```

Master Key URI

If the keyset is wrapped, the master key URI must be made available to Stream:

RPM

Edit your `/etc/default/stream` file and add the variable `STREAM_TINK_MASTER_KEY_URI` with your master-key-uri value.

```
STREAM_TINK_MASTER_KEY_URI=<master key URI>
```

Debian

Edit your `/etc/default/stream` file and add the variable `STREAM_TINK_MASTER_KEY_URI` with your master-key-uri value.

```
STREAM_TINK_MASTER_KEY_URI=<master key URI>
```

Kubernetes

Set `stream.secret.manager.keyset.master-key-uri` using the `extraConfig` value in your Helm `values.yaml` file:

```
extraConfig: |
  stream.secret.manager.keyset.master-key-uri = "<master key URI>"
```

Credentials

If the keyset is wrapped using a KMS, the master key uses credentials that must be made available to Stream. If your credentials are at the standard path for the KMS SDK, it will be found automatically. Otherwise, follow the steps below:

RPM

Edit your `/etc/default/stream` file and add the variable `STREAM_TINK_CREDENTIALS_PATH` with the path to your credentials file.

```
STREAM_TINK_CREDENTIALS_PATH=<path to credentials>
```

Then set the appropriate ownership and permissions:

```
chown stream:stream <path to credentials>
chmod 660 <path to credentials>
```

Debian

Edit your `/etc/default/stream` file and add the variable `STREAM_TINK_CREDENTIALS_PATH` with the path to your credentials file.

```
STREAM_TINK_CREDENTIALS_PATH=<path to credentials>
```

Then set the appropriate ownership and permissions:

```
chown stream:stream <path to credentials>
chmod 660 <path to credentials>
```

Kubernetes

Mount the credentials file into the container and set `stream.secret.manager.keyset.credentials-path` using `extraConfig` in your `values.yaml` file:

```
extraConfig: |
  stream.secret.manager.keyset.credentials-path = "/mnt/sample/path"
```

Key rotation

To rotate the keyset, add a new key and promote it as the primary key. The previous keys must stay `ENABLED` so that the existing secrets can still be decrypted.

Add a new key to the keyset:

RHEL

```
/opt/evertrust/tinkey/bin/tinkey add-key --key-template AES256_GCM --in
stream.keyset --out stream.keyset.new
```

Debian

```
/opt/evertrust/tinkey/bin/tinkey add-key --key-template AES256_GCM --in
stream.keyset --out stream.keyset.new
```

Docker

```
docker run registry.evertrust.io/tinkey add-key --key-template AES256_GCM --in
```

```
stream.keyset --out stream.keyset.new
```

List the keys of the new keyset to retrieve the ID of the key that was just added:

RHEL

```
/opt/evertrust/tinkey/bin/tinkey list-keyset --in stream.keyset.new
```

Debian

```
/opt/evertrust/tinkey/bin/tinkey list-keyset --in stream.keyset.new
```

Docker

```
docker run registry.evertrust.io/tinkey list-keyset --in stream.keyset.new
```

Promote this key as the primary key:

RHEL

```
/opt/evertrust/tinkey/bin/tinkey promote-key --key-id <new key ID> --in
stream.keyset.new --out stream.keyset.rotated
```

Debian

```
/opt/evertrust/tinkey/bin/tinkey promote-key --key-id <new key ID> --in
stream.keyset.new --out stream.keyset.rotated
```

Docker

```
docker run registry.evertrust.io/tinkey promote-key --key-id <new key ID> --in
stream.keyset.new --out stream.keyset.rotated
```



For a wrapped keyset (PKCS#11, AWS KMS or GCP KMS), add the `--master-key-uri` parameter (and `--credentials` if needed) to each of these commands.

The rotated keyset must then be made available to Stream, as described in the Stream configuration section.

The secrets already stored in the database stay encrypted with the previous key. To re-encrypt them with the new primary key, use the Secret Rotator.

1.5. Secret Rotator

This section describes how to rotate the secrets encrypted with a Tink keyset in Horizon and Stream

Installation

RHEL



Secret rotator requires java 17 or 21 installed

Installation from the EverTrust repository

Create a `/etc/yum.repos.d/evertrust-secret-rotator.repo` file containing the EverTrust repository info:

```
[evertrust-secret-rotator]
enabled=1
name=Secret-rotator Repository
baseurl=https://repo.evertrust.io/repository/evertrust-secret-rotator-rpm/
gpgcheck=1
gpgkey=https://evertrust.io/.well-known/rpm/gpg.pub
username=<username>
password=<password>
```

Replace `<username>` and `<password>` with the credentials you were provided.

Make sure the Evertrust GPG key is trusted:

```
# rpm --import https://evertrust.io/.well-known/rpm/gpg.pub
```

You can then run the following to install the latest Secret-rotator version:

```
# yum install evertrust-secret-rotator
```

To prevent unattended upgrades when running yum update, you should pin the Secret-rotator version by adding

```
exclude=evertrust-secret-rotator
```

at the end of the `/etc/yum.repos.d/evertrust-secret-rotator.repo` file after installing Secret-rotator.

Installing from RPM

Download the latest RPM for Secret-rotator on the Official EVERTRUST repository.

Upload the file `evertrust-secret-rotator-<latest>.noarch.rpm` to the server;

Access the server with an account with administrative privileges;

Install the Secret-rotator package with the following command:

```
# yum localinstall /root/evertrust-secret-rotator-<latest>.noarch.rpm
```

If you wish to verify the signature of the RPM package, the EVERTRUST key can be added to your trusted keys using the following command:

```
# rpm --import https://evertrust.io/.well-known/rpm/gpg.pub
```

The signature can then be verified using the following command:

```
# rpm -K /root/evertrust-secret-rotator-<latest>.noarch.rpm
```

Debian



Secret rotator requires java 17 or 21 installed

Installation from the EverTrust repository

If you haven't already, to add the EVERTRUST repository to your APT repositories, run the following commands:

1. Install the required tools (`gpg`)

```
# sudo apt install gnupg
```

2. Download and install the EVERTRUST GPG key

```
# curl https://evertrust.io/.well-known/apt/gpg.pub | sudo gpg -o
/usr/share/keyrings/evertrust.gpg --dearmor
```

3. Add the repository

```
# echo "deb [ arch=all,amd64 signed-by=/usr/share/keyrings/evertrust.gpg ]
https://repo.evertrust.io/repository/apt all main" | sudo tee
/etc/apt/sources.list.d/evertrust.list
```

Once the repository has been added, authentication to it must be provided. To do so, edit the `/etc/apt/auth.conf` file and add the following lines:

```
machine repo.evertrust.io
login <your EVERTRUST login>
password <your EVERTRUST password>
```

Once the repository has been added, run the following command to update the APT repository list.

```
# sudo apt update
```

You can then run the following command to install the latest Secret-rotator version:

```
# sudo apt install evertrust-secret-rotator
```

To prevent unattended upgrades when running `apt upgrade`, you should pin the Secret-rotator version by creating a `/etc/apt/preferences.d/evertrust-secret-rotator` file:

```
Package: evertrust-secret-rotator
Pin: version <installed-version>
Pin-Priority: 1001
```

Installing from DEB

Download the latest DEB for Secret-rotator on the Official EVERTRUST repository.

Upload the file `evertrust-secret-rotator_<latest>_all.deb` to the server;

Access the server with an account with administrative privileges;

Install the Secret-rotator package with the following command:

```
# apt install /root/evertrust-secret-rotator_<latest>_all.deb
```

Docker

The Secret rotator utility is available at:

```
registry.evertrust.io/evertrust-secret-rotator
```

1.6. Monitoring

Healthchecks

Liveness check

The liveness check is available on the `/alive` route of the pekko management port (7626 by default).

It checks that the pekko cluster is operational and performs a ping on the mongo database.

Readiness check

The readiness check is available on the `/ready` route of the pekko management port (7626 by default).

It checks that the pekko cluster is operational and verifies that the instance has been bootstrapped.



For RPM/DEB configuration, this check is proxied by the default NGINX configuration, and available on `/ready`

Metrics

Stream can expose Prometheus metrics to monitor key KPIs and health infos about the instance.

Enabling metrics

RPM

To enable basic Prometheus metrics on port 9095, the following configuration must be applied (following this guide):

```
kamon {
  modules {
    prometheus-reporter.enabled = yes
    apm-reporter.enabled = no
    host-metrics.enabled = no
    jvm-metrics.enabled = no
  }

  prometheus {
    include-environment-tags = true
    embedded-server {
      hostname = 0.0.0.0
      port = 9095
    }
  }
}

stream {
  metrics.enabled = true
```

```
}
```

Debian

To enable basic Prometheus metrics on port 9095, the following configuration must be applied (following this guide):

```
kamon {
  modules {
    prometheus-reporter.enabled = yes
    apm-reporter.enabled = no
    host-metrics.enabled = no
    jvm-metrics.enabled = no
  }

  prometheus {
    include-environment-tags = true
    embedded-server {
      hostname = 0.0.0.0
      port = 9095
    }
  }
}

stream {
  metrics.enabled = true
}
```

Kubernetes

To enable metrics on port 9095, add the following to your `values.yaml` file:

```
metrics:
  enabled: true
  port: 9095
```

Exposed metrics

Exposed metrics are basic clustering metrics, and Stream-specific metrics such as:

- License expiration information
- Stream version
- Scala version
- CRL expiration
- CRL generation
- Signer (CA/TSA/OCSP) expiration

- Keystore status
- Credentials expiration
- Last user activity



Additional metrics configuration such as refresh intervals can be found on the configuration reference page.

1.7. Troubleshooting

Stream Doctor

Stream doctor is a tool that performs checks on your Stream installation as well as its dependencies to ensure that everything is configured properly. Note that the tool requires root permissions to run.

Checks performed

At the moment, Stream doctor checks for :

OS checks

- Checks for installed Stream version, MongoDB version, Java version, Nginx Version and OS version.
  - If the OS is a RedHat distribution, checks for RedHat subscription
  - If Mongo is not installed locally, it notices it as an information log
- Checks for **SELinux**'s configuration
  - If selinux is disabled nothing has to be checked
  - If selinux is enforced checks the `httpd_can_network_connect` sebool value
- Checks for the status of the necessary services: `mongod`, `nginx` and `stream`.
- Checks how long the **stream service** has been running for.
- Checks if there is an **NTP service** active on the machine and checks if the system clock is synchronized with the NTP service.

Config checks

- Checks for existence and permissions of the **configuration** file: the permissions are expected to be at least 640 and the file is supposed to belong to stream:stream.
- Checks for existence and permissions of the **licence** file: the permissions are expected to be at least 640 and the file is supposed to belong to stream:stream.
- Checks for existence and permissions of the **keyset** file: the permissions are expected to be exactly 600 and the file is supposed to belong to stream:stream.
- Checks for existence and permissions of the Stream directory (default : /opt/stream) : the

permission is expected to be at least 755

- Checks for the existence of the **symbolic link** for **nginx configuration** and runs an **nginx -t** test.
- Retrieves the **Java heap size parameters** that were set for Stream and informs the user if the default ones are used (min = 2048 and max = 3072).
- Retrieves the **Stream DNS hostname** and raises an error if it has not been set.
- Retrieves the **MongoDB URI** (throws a warning if MongoDB is running on localhost; throws an error if MongoDB is running on an external instance but the *authSource=admin* parameter is missing from the URI).
- Parses the **licence file** to retrieve its expiration date.
- Checks for the existence of the file containing the initial administrator password and throws a warning if that file still exists (displays the password too)

Network checks

- Runs a **MongoDB ping** on the URI, then checks for the database used in the URI (throws a warning if the database used is not called *stream*; throws an error if no database is specified in the URI).
- Checks for **Pekko High Availability** settings: if no node hostname is set up, skips the remaining HA checks. If 2 nodes are set up, retrieves which node is running the doctor and checks for the other node. If 3 nodes are set up, retrieves which node is running the doctor and checks for the other 2 nodes. The check runs as:
  - if *curl* is installed, runs a *curl* request on the Node hostname at *alive* on the management port (default is 7626), and if alive runs another *curl* request on the Node hostname at */ready* on the management port. Both requests should return HTTP/200 if ok, 000 otherwise.
  - if *curl* is not installed, uses the built-in Linux TCP socket to run TCP SYN checks on both the HA communication port (default is 25520) and the management port (default is 7626) on the Node hostname.
- Checks for **firewall configuration**. Currently only supports *firewalld* (RHEL) and a netstat test.
  - The **netstat part** will run a *netstat* command to check if the JVM listening socket is active (listening on port 9000). If *netstat* is not installed, it will skip this test.
  - The **firewalld part** will check if the HTTP and HTTPS services are opened in the firewall and if it detected a HA configuration, it will check if the HA ports (both of them) are allowed through the firewalld. If *firewalld* is not installed or not active, it will skip this test.
- Checks if IPv6 is active on each network interface and raises a warning if it is the case (with the interface name).

TLS checks

- Checks for existence and permissions of the **Stream server certificate** file: the permissions are expected to be at least 640 and the file is supposed to belong to the nginx group.
- Parses the **Stream server certificate** file: it should be constituted of the actual TLS server certificate first, then of every certificate of the trust chain (order being leaf to root). It throws a

warning if the certificate is self-signed or raises an error if the trust chain has not been imported. It otherwise tries to reconstitute the certificate trust chain via the *openssl verify* command, and throws an error if it cannot.

- Parses the **Stream server certificate** file and checks if the **Stream hostname** is present in the **SAN DNS names** of the certificate, throws an error if it is not there.

Log packing option

If the Stream doctor is launched with the *-l option*, it will pack the logs of the last 7 days (in */opt/stream/var/log*) as well as the startup logs (the */var/log/stream/stream.log* file) and create a tar archive.

The *-l option* accepts an optional parameter that should be an integer (1-99) and will pack the logs of the last n days instead, as well as the startup logs.

Note that the **Stream doctor** will still perform all of its check; the log packing is done at the very end of the program.

Example of call to pack the logs of the last 7 days :

```
# /opt/stream/sbin/stream-doctor -l
```

Example of call to pack the logs of the last 30 days :

```
# /opt/stream/sbin/stream-doctor -l 30
```

Saving the doctor's output

If the Stream doctor is launched with the *-o option*, it will perform all of its checks and save the output in the specified file instead of displaying it into the stdout (default is the commandline interface).

If you use the option, you must provide a filepath in a writable directory.

Example of call to save the output in a file named *stream-doctor.out* instead of the stdout :

```
# /opt/stream/sbin/stream-doctor -o stream-doctor.out
```

Direct fixes

The Stream doctor is able to fix the following issues directly by itself if you use the --fix flag with the script:

- If the application secrets (play secret and event seal secret) have not been changed, the doctor will generate random application secrets and provide them to Stream directly (requires you to manually restart Stream afterwards);
- If firewalld is not allowing HTTP and HTTPS traffic, the doctor will change the firewall settings to allow **both** protocols and then restart the firewall by itself;
- If some permissions for the configuration file, the license file or the keyset file are not what they should be, the doctor will change these permissions (file owner and rwx permissions) to be what they should.

Help menu

To display Stream doctor's help menu, use the *-h option*.

1.8. Advanced configuration

Some technical configurations can be applied to an instance directly in its configuration file. This should be used carefully as it may cause things to break.

Injecting advanced configuration

RPM

On VMs, you have access to the `/opt/stream/etc/conf.d/stream-extra.conf` file. For each parameter you wish to override, create a newline and use the following syntax:

```
<parameter>=<value>
```

As an example, if you want to modify the file extension that DER certificates will have when sent as email attachments and set it to CRT, you need to add:

```
stream.notification.mail.attachment.extension.der="crt"
```

After modifying the file, restart the Stream service:

```
# systemctl restart stream
```



One added line means one modified option, you need to add as many lines at the end of the file as there are values that you want to override.

Debian

On VMs, you have access to the `/opt/stream/etc/conf.d/stream-extra.conf` file. For each parameter you wish to override, create a newline and use the following syntax:

```
<parameter>=<value>
```

As an example, if you want to modify the file extension that DER certificates will have when sent as email attachments and set it to CRT, you need to add:

```
stream.notification.mail.attachment.extension.der="crt"
```

After modifying the file, restart the Stream service:

```
# systemctl restart stream
```



One added line means one modified option, you need to add as many lines at the end of the file as there are values that you want to override.

Kubernetes

The Stream container provides a bundled `application.conf` file that is mostly configured through environment variables. To modify low-level behavior of Stream that is not accessible through an environment variable, use the `extraConfig` value in your `values.yaml` file to update specific settings:

```
extraConfig: |
    stream {
      metrics.enabled = true
    }
```

Extra configurations are appended at the end of the config file, overriding any previously set config value.

Available settings



Parameter `stream.security.http.headers.xapi.idprov` was deleted.



Parameter `stream.security.http.headers.xapi.key` was deleted.



Parameter `stream.security.http.headers.xapi.id` was deleted.



Parameter `stream.security.http.headers.xid` was deleted.



Parameter `stream.trustchain.ca.online.root.operational` was deleted.



Parameter `stream.trustchain.ca.online.root.non_operational` was deleted.



Parameter `stream.trustchain.ca.online.subordinate.operational` was deleted.



Parameter `stream.trustchain.ca.offline.root.non_operational` was deleted.



Parameter `stream.crl.manager.timeout` was deleted.



Parameter `stream.ocsp.manager.timeout` was deleted.

Parameter `stream.timestamping.manager.timeout` was deleted.

Parameter `stream.crl.queue.size` was deleted.

Bootstrap Configuration

stream.bootstrap.administrator.name

```
stream.bootstrap.administrator.name = "administrator"
```

Default administrator account name

stream.bootstrap.administrator.display-name

```
stream.bootstrap.administrator.display-name = "Stream Administrator"
```

Default administrator account display name

This parameter replaces `stream.bootstrap.administrator.display.name`. Please modify your configuration accordingly

stream.bootstrap.administrator.password.path

```
stream.bootstrap.administrator.password.path = "var/run/adminPassword"
```

Relative path of the file where the initial admin password should be stored into

stream.bootstrap.local.identity.provider

```
stream.bootstrap.local.identity.provider = "local"
```

Length (in bytes) of the initial admin password

Default administrator account identity provider to use

stream.bootstrap.timeout

```
stream.bootstrap.timeout = "1m"
```

Duration after which the bootstrap of Stream times out

CRL Configuration

stream.crl.sync.interval

```
stream.crl.sync.interval = "15m"
```

Interval at which CRL synchronization occurs

stream.crl.generation.queue.parallelism

```
stream.crl.generation.queue.parallelism = 5
```

Maximum number of CRL generation jobs running concurrently across all Certificate Authorities. Generation for a single Certificate Authority is always serialized (one job at a time), this only caps the total number of distinct Certificate Authorities being processed in parallel.

stream.crl.generation.queue.size

```
stream.crl.generation.queue.size = 100
```

Number of CRL generation requests that can be buffered before new ones are dropped.

stream.crl.cache.max-age.mode

```
stream.crl.cache.max-age.mode = "1s"
```

How to set max-age cache directive on crl fetch: one of 'disabled', 'nextrefresh' or a duration

stream.crl.cache.max-age.default

```
stream.crl.cache.max-age.default = "5m"
```

Default max-age duration in 'nextrefresh' mode when the CRL has no next refresh planned

stream.crl.upload.max-size

```
stream.crl.upload.max-size = "20m"
```

Max allowed size on applicative side for CRL uploads

Certificate authentication

stream.security.http.headers.certificate

```
stream.security.http.headers.certificate = null
```

Name of the HTTP header containing the certificate

stream.security.authentication.enforce-x509

```
stream.security.authentication.enforce-x509 = false
```

Allow only certificate authentication

Event Configuration

stream.event.ttl

```
stream.event.ttl = null
```

Time to live of the events. If not set, events never expire

stream.event.chainsign

```
stream.event.chainsign = true
```

Specify whether to chain and sign the Stream events to ensure they haven't been tampered with

stream.event.seal.algorithm

```
stream.event.seal.algorithm = "HS512"
```

Algorithm to use to hash the signature of the events in Stream (other possible values are "HS384" and "HS256")

stream.event.seal.secret

```
stream.event.seal.secret = null
```

Secret to seal the events with

stream.event.ignore-unsealed-pending

```
stream.event.ignore-unsealed-pending = false
```

Do not throw an error if pending events are unsealed

stream.event.disable-stacktrace

```
stream.event.disable-stacktrace = false
```

Enable to remove stacktraces from Stream events

stream.event.timeout

```
stream.event.timeout = "30s"
```

Duration after which the event manager times out when trying to retrieve the last signed event in the database

stream.event.manager.interval

```
stream.event.manager.interval = "5s"
```

How often will the Event Manager actor check in the database if new a new event appeared to sign it and display it in the "Events" section of Stream

General

stream.security.trustmanager.enforce-serverauth

```
stream.security.trustmanager.enforce-serverauth = false
```

If set to true, enforces the use of the serverAuth EKU in the server authentication certificates (when Stream accesses a service through TLS)



This parameter replaces `stream.security.trustmanager.enforce_serverauth`. Please modify your configuration accordingly

stream.security.trustmanager.timeout

```
stream.security.trustmanager.timeout = "10s"
```

Timeout to check trust status of certificates



This parameter replaces `stream.trust.manager.timeout`. Please modify your configuration accordingly

stream.security.trustmanager.cache.expire-after-access.external

```
stream.security.trustmanager.cache.expire-after-access.external = "30d"
```

Time after which an entry in the CRL cache expires for external CAs



This parameter replaces `stream.trust.manager.cache.external.expireafteraccess`. Please modify your configuration accordingly

stream.security.trustmanager.cache.expire-after-access.managed

```
stream.security.trustmanager.cache.expire-after-access.managed = "5m"
```

Time after which an entry in the CRL cache expires for managed CAs



This parameter replaces `stream.trust.manager.cache.managed.expireafteraccess`. Please modify your configuration accordingly

stream.security.trustmanager.crl-info.interval

```
stream.security.trustmanager.crl-info.interval = "5m"
```

Interval at which CRL Info are synchronized in trust manager

stream.security.manager.timeout

```
stream.security.manager.timeout = "10s"
```

Duration after which the security manager times out when trying to authenticate a principal with its session

stream.security.principal.password.length

```
stream.security.principal.password.length = 42
```

Local accounts password length



This parameter replaces `stream.account.secret.length`. Please modify your configuration accordingly

stream.keystore.timeout

```
stream.keystore.timeout = "1m"
```

How long the authentication cache lasts

Timeout for operations using keystores (generating CSR, listing keys, etc ..)

stream.keystore.pkcs11.reload.delay

```
stream.keystore.pkcs11.reload.delay = "5s"
```

Delay when reloading pkcs11 keystores after an error

stream.keystore.healthcheck.interval

```
stream.keystore.healthcheck.interval = "5m"
```

Interval at which keystore status is checked

stream.keystore.required-for-readiness

```
stream.keystore.required-for-readiness = []
```

List of names of keystores that are required to consider the instance ready

stream.queue.timeout

```
stream.queue.timeout = "5s"
```

Timeout to register the queues in actors

stream.queue.parallelism

```
stream.queue.parallelism = 5
```

Number of parallel requests (enrollment, revocation, ocsp, timestamping...) on the default queue



This parameter replaces `stream.queue.default.parallelism`. Please modify your

configuration accordingly

stream.queue.size

```
stream.queue.size = 100
```

Number of requests (enrollment, revocation, ocsp, timestamping, crl, krl) that can be queued on the default queue



This parameter replaces `stream.queue.default.size,stream.crl.queue.size`. Please modify your configuration accordingly

stream.metrics.enabled

```
stream.metrics.enabled = false
```

Enable advanced metrics for collection

stream.metrics.intervals.short

```
stream.metrics.intervals.short = "30s"
```

Interval at which short lived metrics are computed

stream.metrics.intervals.long

```
stream.metrics.intervals.long = "5m"
```

Interval at which background metrics are computed

stream.trigger.timeout

```
stream.trigger.timeout = "1m"
```

Timeout for registering the triggers in actors

stream.ntp.client.timeout

```
stream.ntp.client.timeout = "1m"
```

Timeout for registering the NTP Clients in actors

stream.system.monitor.timeout

```
stream.system.monitor.timeout = "1m"
```

Timeout for the system monitor loading



This parameter replaces `stream.system.configuration.timeout`. Please modify your configuration accordingly

stream.sql.max-recursion-depth

```
stream.sql.max-recursion-depth = 5
```

Maximum recursion allowed for the SQL (Stream Query Language) queries

HTTP Headers Configuration

stream.security.http.headers.enforce-connection-close

```
stream.security.http.headers.enforce-connection-close = true
```

Defines whether HTTP connections should remain open



This parameter replaces `stream.http.header.enforce_connection_close`. Please modify your configuration accordingly

stream.security.http.headers.real-ip

```
stream.security.http.headers.real-ip = "X-Real-IP"
```

Name of the HTTP header to use as Real IP



This parameter replaces `stream.http.header.realip`. Please modify your configuration accordingly

KRL Configuration

stream.krl.sync.interval

```
stream.krl.sync.interval = "15m"
```

Interval at which KRL synchronization occurs

stream.krl.generation.queue.parallelism

```
stream.krl.generation.queue.parallelism = 5
```

Maximum number of KRL generation jobs running concurrently across all Certificate Authorities. Generation for a single Certificate Authority is always serialized (one job at a time), this only caps the total number of distinct Certificate Authorities being processed in parallel.

stream.krl.generation.queue.size

```
stream.krl.generation.queue.size = 100
```

Number of KRL generation requests that can be buffered before new ones are dropped.

stream.krl.cache.max-age.mode

```
stream.krl.cache.max-age.mode = "1s"
```

How to set max-age cache directive on krl fetch: one of 'disabled', 'nextrefresh' or a duration

stream.krl.cache.max-age.default

```
stream.krl.cache.max-age.default = "5m"
```

Default max-age duration in 'nextrefresh' mode when the KRL has no next refresh planned

Keyset configuration

stream.secret.manager.keyset.path

```
stream.secret.manager.keyset.path = "etc/stream.keyset"
```

Path to the keyset for secrets encryption

stream.secret.manager.keyset.master-key-uri

```
stream.secret.manager.keyset.master-key-uri = null
```

Master key URI to encrypt the keyset with

OCSP Configuration

stream.ocsp.timeout

```
stream.ocsp.timeout = "1m"
```

Timeout for processing OCSP requests and starting OCSP actors

stream.ocsp.request.max-size

```
stream.ocsp.request.max-size = "8k"
```

Max allowed size for OCSP requests



This parameter replaces `stream.ocsp.request.maxsize`. Please modify your configuration accordingly

stream.ocsp.default-next-update-delay

```
stream.ocsp.default-next-update-delay = "5m"
```

Default time for OCSP response next update when no crl refresh is available



This parameter replaces `stream.ocsp.default.next_update_delay`. Please modify your configuration accordingly

OpenID Configuration

stream.openid.state-separator

```
stream.openid.state-separator = "#"
```

Separator character of the OpenID state



This parameter replaces `stream.security.identity.provider.openid.state.separator`. Please modify your configuration accordingly

stream.openid.nonce.size

```
stream.openid.nonce.size = 32
```

Size (in bytes) of the challenge stored in the nonce



This parameter replaces `stream.security.identity.provider.openid.nonce.size`. Please modify your configuration accordingly

stream.openid.nonce.ttl

```
stream.openid.nonce.ttl = "1m"
```

Duration for which a nonce stays in Stream before being removed



This parameter replaces `stream.security.identity.provider.openid.nonce.ttl`. Please modify your configuration accordingly

stream.openid.callback-scheme

```
stream.openid.callback-scheme = "https"
```

Scheme to use for the callback

SSH Configuration

stream.ssh.ca.timeout

```
stream.ssh.ca.timeout = "1m"
```

Timeout for registering the SSH Certificate Authorities in actors

The Timeout of SSH CA actions

Search Configuration

stream.security.principal.search.page.default-size

```
stream.security.principal.search.page.default-size = 50
```

How many elements to retrieve in a security principals search query if no pageSize has been specified



This parameter replaces `stream.security.principal.search.page.default_size`. Please modify your configuration accordingly

stream.security.principal.search.page.max-size

```
stream.security.principal.search.page.max-size = null
```

How big can the pageSize parameter be in a security principals search query (Must be a positive integer)



This parameter replaces `stream.security.principal.search.page.max_size`. Please modify your configuration accordingly

stream.event.search.page.default-size

```
stream.event.search.page.default-size = 50
```

How many elements to retrieve in an event search query if no pageSize has been specified



This parameter replaces `stream.event.search.page.default_size`. Please modify your configuration accordingly

stream.event.search.page.max-size

```
stream.event.search.page.max-size = null
```

How big can the pageSize parameter be in an event search query (Must be a positive integer)



This parameter replaces `stream.event.search.page.max_size`. Please modify your configuration accordingly

stream.x509.certificate.search.page.default-size

```
stream.x509.certificate.search.page.default-size = 50
```

How many elements to retrieve in a X509 certificate search query if no pageSize has been specified



This parameter replaces `stream.certificate.search.page.default_size`. Please modify your configuration accordingly

stream.x509.certificate.search.page.max-size

```
stream.x509.certificate.search.page.max-size = null
```

How big can the pageSize parameter be in a X509 certificate search query (Must be a positive

integer)



This parameter replaces `stream.certificate.search.page.max_size`. Please modify your configuration accordingly

stream.ssh.certificate.search.page.default-size

```
stream.ssh.certificate.search.page.default-size = 50
```

How many elements to retrieve in a SSH certificate search query if no pageSize has been specified

stream.ssh.certificate.search.page.max-size

```
stream.ssh.certificate.search.page.max-size = null
```

How big can the pageSize parameter be in a SSH certificate search query (Must be a positive integer)

TSA Configuration

stream.timestamping.timeout

```
stream.timestamping.timeout = "1m"
```

Timeout to register signers and process responses

stream.timestamping.authority.timeout

```
stream.timestamping.authority.timeout = "1m"
```

Timeout to register timestamping authorities in actors

stream.timestamping.request.max-size

```
stream.timestamping.request.max-size = "8k"
```

Max allowed size for timestamping requests



This parameter replaces `stream.timestamping.request.maxsize`. Please modify your configuration accordingly

X509 Configuration

stream.x509.ca.timeout

```
stream.x509.ca.timeout = "1m"
```

Timeout for registering the X509 Certificate Authorities in actors

The Timeout of X509 CA actions



This parameter replaces `stream.ca.timeout`. Please modify your configuration accordingly

1.9. Upgrade

Upgrade the Stream installation

The first step in the upgrade procedure is to upgrade the Stream component itself.

RPM

Installation from the EverTrust repository

If you installed Stream using our repository (as described in the installation section), you should:

- Unpin the Stream version by commenting out any line excluding the `stream` package in the `/etc/yum.repos.d/stream.repo` repository file :

```
[stream]
enabled=1
name=Stream Repository
# exclude=stream
```

- Run `yum update stream`

Don't forget to pin the version again by uncommenting the line that was previously commented.

Installing from RPM

You must retrieve the latest Stream RPM from the EverTrust repository manually using the user credentials you were provided.

Connect to the server with an account with administrative privileges;

Install the Stream package with the following command:

```
# yum localinstall stream-1.0.x-1.x86_64.rpm
```

Debian

Installation from the EverTrust repository

If you installed Stream using our repository (as described in the installation section), you should:

- Unpin the Stream version by removing the pinned version for the `stream` package in the `/etc/apt/preferences.d/stream` repository file :

```
Package: stream
Pin: version <installed-version>
Pin-Priority: 1001
```

- Run `apt update stream`

Don't forget to pin the version again.

Installing from DEB

You must retrieve the latest Stream DEB from the EverTrust repository manually using the user credentials you were provided.

Connect to the server with an account with administrative privileges;

Install the Stream package with the following command:

```
# apt install stream-1.0.x_amd64.deb
```

Kubernetes

When upgrading Stream, you'll need to pull the latest version of the chart:

```
$ helm repo update evertrust
```

Verify that you now have the latest version of Stream (through the App version column):

```
$ helm search repo evertrust/stream
 NAME                            CHART VERSION   APP VERSION DESCRIPTION
 evertrust/stream          0.9.3           2.2.0       EverTrust Stream Helm
chart
```

Launch an upgrade by specifying the new version of the chart through the `--version` flag in your command:

```
$ helm upgrade <stream> evertrust/stream \
  --values override-values.yaml \
  --version 0.9.3
```



We recommended that you only change values you need to customize in your `values.yml` file to ensure smooth upgrading.

Docker

Pull the new image from the repository.

Upgrade the database schema

Some Stream versions require that you run migration scripts against your database. Stream comes bundled with an `stream-upgrade` script that handles this migration logic.

Therefore, after each upgrade, you should run `stream-upgrade` to check whether new migrations should be run.

RPM

Connect to the server with an account with administrative privileges;

Run the following command:

```
# /opt/stream/sbin/stream-upgrade -t <target version>
```

In most cases, `stream-upgrade` can detect the version you're upgrading from by checking the database. if the source version is not automatically detected, you will encounter the following error:

```
*** Unable to infer the source version from your database. Specify it
explicitly with the -s flag. ***
```

You'll have to explicitly tell `stream-upgrade` which version you are upgrading from. To do that, simply set the source version explicitly with the -s flag :

```
# /opt/stream/sbin/stream-upgrade -t <target version> -s <source version>
```

Similarly, `stream-upgrade` will try to use the MongoDB URI that was configured by the Stream configuration utility. If it fails to auto-detect your database URI or you wish to migrate another database, specify the URI explicitly using the `-m` flag:

```
# /opt/stream/sbin/stream-upgrade -t <target version> -m "<MongoDB connection
```

```
string>"
```



The upgrade script requires the mongo shell MongoSH to connect to your database (`mongosh`). If this client is not installed on the host where Stream is running, consider installing the standalone `mongosh` client or running the upgrade script from another host that has access to the database.

Debian

Connect to the server with an account with administrative privileges;

Run the following command:

```
# /opt/stream/sbin/stream-upgrade -t <target version>
```

In most cases, `stream-upgrade` can detect the version you're upgrading from by checking the database. if the source version is not automatically detected, you will encounter the following error:

```
*** Unable to infer the source version from your database. Specify it
explicitly with the -s flag. ***
```

You'll have to explicitly tell `stream-upgrade` which version you are upgrading from. To do that, simply set the source version explicitly with the -s flag :

```
# /opt/stream/sbin/stream-upgrade -t <target version> -s <source version>
```

Similarly, `stream-upgrade` will try to use the MongoDB URI that was configured by the Stream configuration utility. If it fails to auto-detect your database URI or you wish to migrate another database, specify the URI explicitly using the `-m` flag:

```
# /opt/stream/sbin/stream-upgrade -t <target version> -m "<MongoDB connection
string>"
```



The upgrade script requires the mongo shell MongoSH to connect to your database (`mongosh`). If this client is not installed on the host where Stream is running, consider installing the standalone `mongosh` client or running the upgrade script from another host that has access to the database.

Kubernetes

The chart will automatically create a `Job` that runs an upgrade script when it detects that the Stream version has changed between two releases. If the upgrade job fails to run, check the job's pod logs. When upgrading from an old version of Stream, you may need to explicitly specify the version you're upgrading from using the `upgrade.from` key.



Before upgrading to specific chart version, thoroughly read any Specific upgrade instructions for your version.

Docker

You can run the script on any computer that has access to your MongoDB database, provided that it has Docker installed:

```
$ docker run -it --rm registry.evertrust.io/stream-upgrade:<target version> \
  -m <mongo uri> \
  -t <target version> \
  -s <source version>
```

Specific upgrade instructions

Architecture-specific instructions

RPM

No specific instructions.

Debian

No specific instructions.

Kubernetes

Upgrading to 1.7.0

- Switching to native Kubernetes leases implementation. CRDs leases aren't used anymore.

Upgrading to 2.0.0

- This version drops support for the Bitnami MongoDB subchart. Instead, a new `temporaryDatabase` key controls whether a temporary MongoDB instance should be created for the duration of the upgrade. To migrate from the Bitnami MongoDB subchart to a temporary instance or an external MongoDB database, you can use the `mongodump` and `mongorestore` utilities.

Docker

No specific instructions.

1.10. Uninstallation



Before uninstalling, please ensure that you have a **proper backup of the Stream component**. Once uninstalled, all Stream data will be **irremediably lost**!



Uninstalling Stream consists in uninstalling:

- The Stream service;
- The MongoDB service;
- The NGINX service.

Uninstallation procedure

RPM

First, uninstall Stream with the following commands:

```
# systemctl stop stream
# yum remove stream
# rm -rf /opt/stream
# rm -rf /var/log/stream
# rm -f /etc/default/stream
```

If NGINX was installed alongside with Stream, you can remove it with the following commands:

```
# systemctl stop nginx
# yum remove nginx
# rm -rf /etc/nginx
# rm -rf /var/log/nginx
```

The same cleanup operation apply to MongoDB, which can be removed with the following commands:

```
# systemctl stop mongod
# rpm -qa | grep -i mongo | xargs rpm -e
# rm -rf /var/log/mongodb
# rm -rf /var/lib/mongodb
```

Debian

First, uninstall Stream with the following commands:

```
# systemctl stop stream
# apt remove stream
# rm -rf /opt/stream
# rm -rf /var/log/stream
# rm -f /etc/default/stream
```

If NGINX was installed alongside with Stream, you can remove it with the following

commands:

```
# systemctl stop nginx
# apt remove nginx
# rm -rf /etc/nginx
# rm -rf /var/log/nginx
```

The same cleanup operation apply to MongoDB, which can be removed with the following commands:

```
# systemctl stop mongod
# apt-get purge "mongodb-org*"
# rm -rf /var/log/mongodb
# rm -rf /var/lib/mongodb
```

Kubernetes

To uninstall Stream from your cluster, simply run:

```
$ helm uninstall stream -n stream
```

This will uninstall Stream. If you installed a local MongoDB instance through the Stream's chart, it will also be uninstalled, meaning you'll lose all data from the instance.



Before uninstalling Stream, if you wish to keep your database, please back up your application secrets (in particular the keyset). Without it, you won't be able to decrypt your database and it will become useless.

Chapter 2. Admin guide

2.1. Introduction

Description

Stream is EverTrust Certificate Authority solution and is powered up by:

- Pekko
- BouncyCastle
- MongoDB
- Kamon
- Play! Framework
- Scala
- NGINX
- Vue.js
- Quasar

This document is specific to Stream version **1.0**, and may apply to follow-up minor releases.

Scope

This document is an administration guide detailing how to configure and operate Stream.

Out of Scope

This document does not describe how to install and bootstrap a Stream instance. Please refer to the installation guide for installation related tasks.

2.2. Managing Certification Authorities

2.2.1. Importing an External Certification Authority

1. Log in to the Stream Administration Interface.

2. Go to **Certification Authorities** > **External CAs** and click on  .

3. You need to provide the X509 CA Certificate, either by pasting it directly into the box or by importing the file. **PEM** and **DER** formats are supported. Then click "Next".

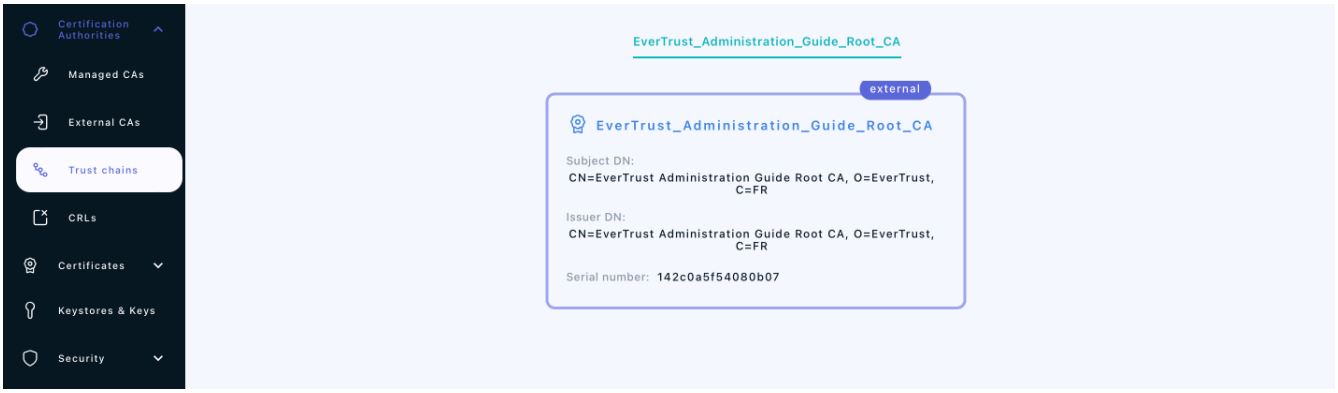
4. In the **Details** tab, check if the details that were parsed from the certificate match those of the CA you wish to import. If it does, click "Next".

5. In the Configuration tab, you can

- Add a **CRL**
- Edit the **Refresh period**
- Edit the **Timeout timer**
- Configure a **proxy**
- Toggle whether the external CA should be trusted for **server authentication** or **client authentication**
- Specify the **Outdated Revocation Status Policy**
- Enable OCSP and configure a **Default OCSP Signer**

6. You can then click the "Import" button in the bottom right corner to import your CA.

If everything was ok, you should see your CA marked as *external* if you go to **Certification Authorities** > **Trust chains**:



2.2.2. Importing an existing Managed Certification Authority

1. Log in to the Stream Administration Interface.

2. Go to **Import existing CA** from the menu on the left

3. Import your CA certificate file or paste the content of the file in the *Copy/paste the certificate* box. If you decide to paste the file's content, don't forget to click the parse button on the right before continuing.

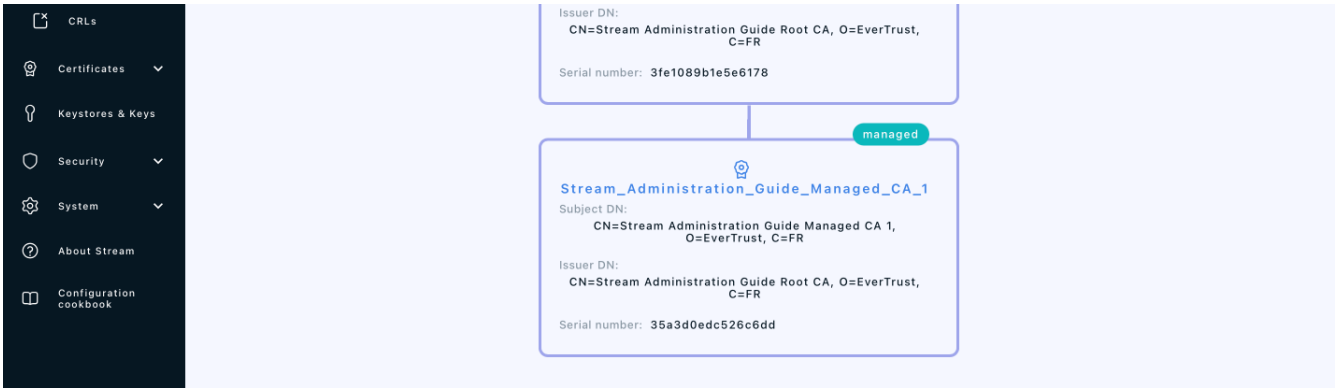
4. Scroll down to the bottom of the page and check the certificate's information. If everything is correct, click "Next".

5. Select the Keystore where your CA's key is stored. If you do not have a keystore set up yet, please refer to the ***Managing Keystores & Keys*** section.

6. Select the key that was used to generate the CA from the selected keystore and click "Next".

7. Upload your CA's CRL file and click "Add".

If everything was ok, you should see your CA marked as *managed* if you go to **Certification Authorities** > **Trust chains**:



2.2.3. Issuing a new Root Certification Authority

1. Log in to the Stream Administration Interface.

2. Go to **Create a new CA** from the menu on the left.

3. Input your CA's **internal name** and manage the **DNs** that you want to add (using the `Add a DN element` button on the bottom left corner) or to remove (using the icon).

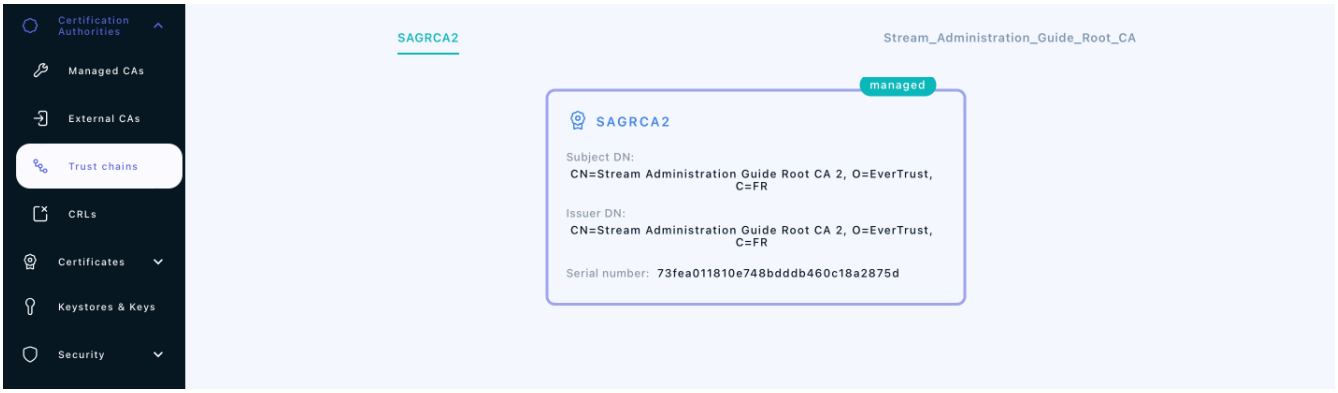
4. Select the **Keystore** that contains the key you want to use to generate this CA, then select the key(s) that you want to use. If you do not have a keystore set up yet, please refer to the ***Managing Keystores & Keys*** section.

5. Select **Selfsigned** as a signing method, and pick the hash algorithm of your choice. Optionally, if you picked a PKCS#11 Keystore and an RSA key, you have the ability to use a PSS signature instead of the classic PKCS#1 one : if you wish to do so, just turn on the toggle. Note that your HSM must support the `CKM_RSA_PKCS_PSS` mechanism.

6. Set the **lifetime** of your CA in days. Optionally, you can set up a **backdate** and a **path length**. Once you are done, click "Add".

7. You can directly configure your CA from this menu, by turning on or off **enrollment**, trusting the CA for **client authentication** or **server authentication**, enabling **OCSP** or **enforcing key unicity**. Once you're satisfied with your settings, click "Add".

If everything was ok, you should see your CA marked as **managed** on a new trust chain if you go to **Certification Authorities** > **Trust chains**:



2.2.4. Issuing a subordinate Certification Authority

Signed locally

1. Log in to the Stream Administration Interface.

2. Go to **Create a new CA** from the menu on the left.

3. Input your CA's **internal name** and manage the **DNs** that you want to add (using the `Add a DN element` button on the bottom left corner) or to remove (using the 🗑 icon).

4. Select the **Keystore** that contains the key you want to use to generate this CA, then select the key(s) that you want to use. If you do not have a keystore set up yet, please refer to the ***Managing Keystores & Keys*** section.

5. Select **Signed with an internal CA** as the signing method.

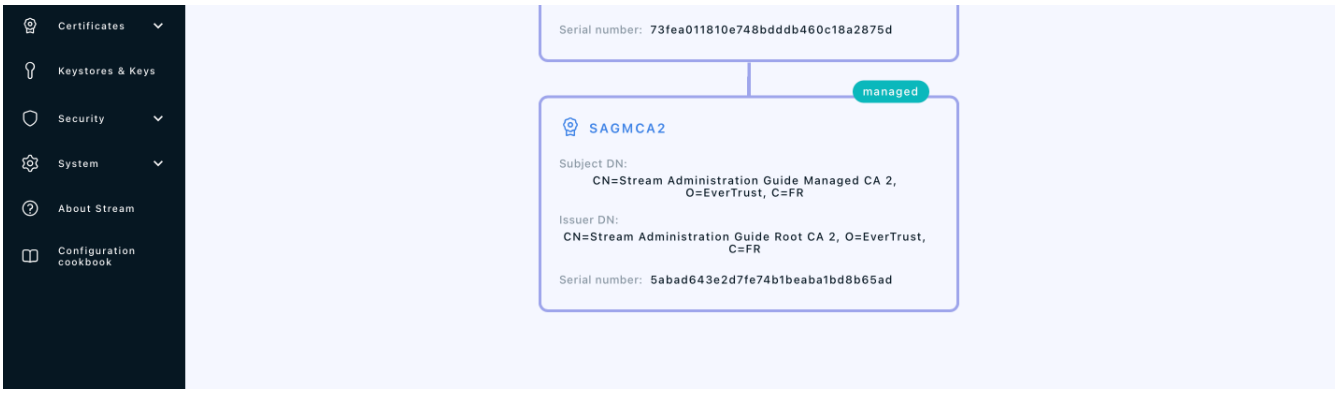
6. Select the **Managed CA** you want to sign the certificate with.

7. Set the **lifetime** or your CA in days. Optionally, you can set up a **backdate** and a **path length**.

8. Optionally, you can set up an **OID Policy**, a **CPS Pointer**, add **CRLDPs** and the CA's **AIA**. Once you are finished with the settings, click "Issue CA".

9. You can directly configure your CA from this menu, by turning on or off **enrollment**, trusting the CA for **client authentication** or **server authentication**, enabling **OCSP** or **enforcing key unicity**. Additionally, if you issued this CA using an RSA key from a PKCS#11 keystore, you can choose to use the PSS signature algorithm instead of the default PKCS#1 one to sign new certificates. To do so, simply turn on the toggle. Note that your HSM must support the `CKM_RSA_PKCS_PSS` mechanism.
Once you're satisfied with your settings, click "Add".

If everything was ok, you should see your CA marked as **managed** on a new trust chain if you go to **Certification Authorities** > **Trust chains**:



Signed externally

1. Log in to the Stream Administration Interface.

2. Go to **Create a new CA** from the menu on the left.

3. Input your CA's **internal name** and manage the **DNs** that you want to add (using the `Add a DN element` button on the bottom left corner) or to remove (using the icon).

4. Select the **Keystore** that contains the key you want to use to generate this CA, then select the key(s) that you want to use. If you do not have a keystore set up yet, please refer to the ***Managing Keystores & Keys*** section.

5. Select **Signed with an external CA** as the signing method.

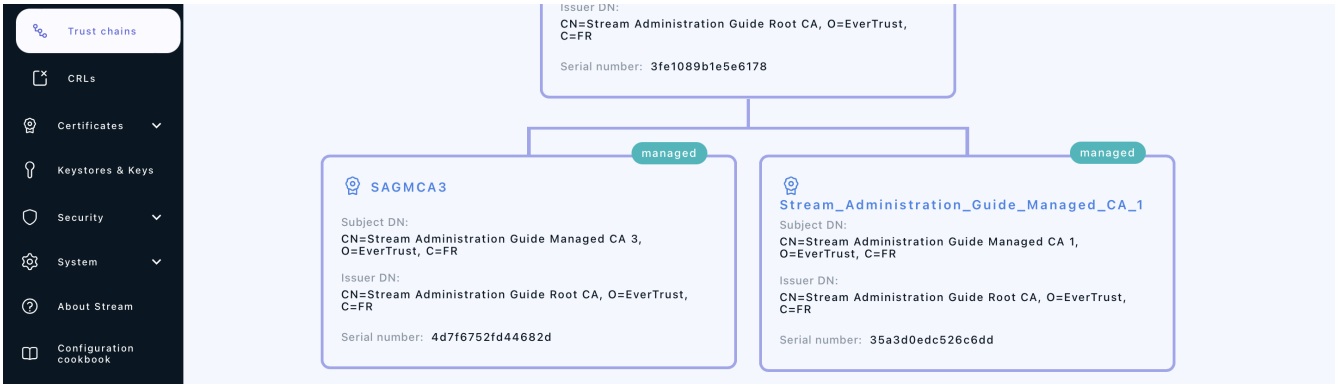
6. Click the link in the **Export** section to download the **CSR** for your CA, then sign it using your external CA and export the signed certificate under PEM or DER format.

7. Upload the signed certificate in the **Import** section.

8. Scroll down to the bottom of the page and check the certificate's information. If everything is correct, click "Next".

9. You can directly configure your CA from this menu, by turning on or off **enrollment**, trusting the CA for **client authentication** or **server authentication**, enabling **OCSP** or **enforcing key unicity**. Once you're satisfied with your settings, click "Add".

If everything was ok, your should see your CA marked as **managed** on a new trust chain if you go to **Certification Authorities** > **Trust chains**:



2.2.5. Note on CRLDP and AIA settings



Regardless of the CA type, the setting "CRLDP" refers to the CRL of the CA you are configuring, and **NOT** the one of the issuing CRL. Same goes for the AIA: you need to specify the certificate of the CA you are configuring, and not the certificate of its issuing CA.

2.2.6. AIA Certificate Issuer

Stream allows you to download the **Certificate** of the **CAs** (external and managed). This is usually used in AIA issuer certificate extension to be able to download the certificate of the issuing Certificate Authority.

The standard download URL format is *http(s)://[stream_url]/aias/CA_internal_name*. This URL can be accessed by anyone without prior authentication, either through HTTP or HTTPS.

You need to specify the **Internal name** of the CA to download its **certificate** and not its **Common Name (CN)**.

The certificate format depends on the request `ACCEPT` header:

- `application/x-pem-file`: returns the certificate in **PEM**
- `application/pkix-cert`: returns the certificate in **DER**
- `application/x-pkcs7-certificates`: returns the certificate in **PKCS7**

If no `ACCEPT` header is specified, return the certificate in **DER**.

The certificate is returned with the following headers:

- `Content-Type`:
  - `application/x-pem-file` for **PEM**
  - `application/pkix-cert` for **DER**
  - `application/x-pkcs7-certificates` for **PKCS7**
- `Content-Disposition`: 'attachment; filename=<ca name>'

2.3. Managing Certificate Revocation

2.3.1. Configuring Certificate Revocation Lists for an External CA

1. Log in to the Stream Administration Interface ;

2. Go to **Certification Authorities** > **External CAs** and click on ⬆ next to the name of the CA you want to import the CRL of ;

3. Select a valid CRL file that has been signed by your CA ;

4. If everything went through correctly, the CRL of that external CA should be available to download from Stream ;

5. Additionally, if you want to push the CRL into a CRL storage, click ✎ on the external CA ;

5.1 In the **Configuration** tab, select one or several previously created external storages from the drop-down menu:

- **On CRL update**: this will be triggered every time a new CRL is uploaded (see step **2**).
- **On CRL sync**: this will trigger every 15 minutes to ensure CRL is up to date on the storage, and push the new one if needed

5.2 Click the **Save** button at the top.

The CRL should now also be pushed in the CRL storage(s) whenever you manually import it into Stream. Note that the CRL will still be accessible from the standard Stream CRLDP.

2.3.2. Configuring Certificate Revocation Lists for a Managed CA

To manage the **CRLs** of a managed CA, you first need to set up a **CRL Policy**:

1. Log in to the Stream Administration Interface.

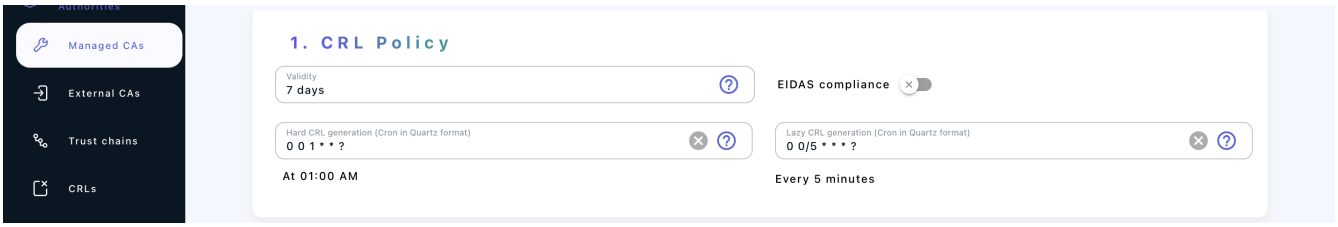
2. Go to **Certification Authorities** > **Managed CAs** and click on ✎ next to the name of the CA you want to edit the CRL policy of.

3. Go under the **CRL/OCSP** tab.

4. First, you need to define the validity period of your CRL, i.e. the period of time while your CRL is considered valid. The countdown starts at the moment the CRL is generated. If you want your CRLs to be valid for a week, you can type 7 **days**.

5. You can then automate the **CRL generation** using either the **Hard CRL generation**, the **Lazy CRL generation** or both of them in combination:

- The **Hard CRL generation** parameter takes a cron expression in Quartz format and generates the CRL every time that cron expression is valid, without any condition. It is recommended to generate the **CRLs** every day. To generate a new **CRL** every day at 1 A.M., the cron expression is: 0 0 1 * * ?
- The **Lazy CRL generation** parameter takes a cron expression in Quartz format and checks if the CRL needs to be updated, i.e. if a certificate has been revoked, since the last CRL generation. If a certificate has been revoked since the last generation then a new CRL will then be generated, otherwise it will do nothing. It is recommended to have a short time span for the lazy generation so that the CRL always stays up to date. To check for possible CRL updates every 5 minutes, the cron expression is: 0 0/5 * * * ?



6. Click the **Save** button at the top of the page.

Now your CRL policy has been configured, and you've been redirected to the Managed CAs page.

You can then generate manually the CA's first CRL using the ↴ button next to the CA's name that you just configured. If you configured the **Hard** or the **Lazy** generation, your CRL will then automatically be updated according to the cron quartz expression you specified.

7. Additionally, if you want to push the CRL into other storages, click ✎ on the managed CA ;

7.1 In the **Configuration** tab, select one or several previously created external storages from the drop-down menu:

- **On CRL generation**: this will be triggered every time a new CRL is generates (manually or via

the configuration at step **5**).

- **On CRL sync**: this will trigger every 15 minutes to ensure CRL is up to date on the storage, and push the new one if needed

7.2 Click the **Save** button at the top.

The CRL should now also be pushed in other storages. Note that the CRL will still be accessible from the standard Stream CRLDP.

2.3.3. Viewing CRLs

1. Log in to the Stream Administration Interface.

2. Go to **Revocation Management** > **CRLs**.

3. You can then see information regarding your CAs' CRLs that are going to be detailed below:



- The **CA** column indicates the name of the CA whose CRL is detailed in the line
- The **Number** column indicates the serial number of the CRL. It starts at 1 for the very first CRL generated and is incremented by 1 at each generation. It is displayed in hexadecimal format.
- The **Last update** column indicates the date and time when the current CRL was generated.
- The **Valid Until** column indicates the date and time when the current CRL will expire. It should be equal to *Last update* + the validity period you set in the **CRL policy** field.
- The **Next refresh** column indicates the date and time when the current CRL will be refreshed. It should be equal to the nearest date matching either cron quartz expression you set in the **CRL policy** field (lazy or hard).
- The **download** ⬇ **button** allows you to download your CRL. It also serves as a CRLDP. For more information about CRLDPs in Stream, please refer to next section.
- The **generate** ⤵ **button** allows you to manually refresh the CRL and generates a new one.
- The **refresh** ⟳ **button** refreshes the information displayed in the tab, in case a generation happened in between. It **does not** refresh the CRLs, only the displayed information.

2.3.4. Downloading CRLs

Stream allows you to download the **CRLs** of the **CAs** it manages. The standard download URL format is *http(s):///[stream_url]/crls/CA_internal_name*. This URL can be accessed by anyone without prior authentication, either through HTTP or HTTPS.

You need to specify the **Internal name** of the CA to download its **CRL** and not its **Common Name (CN)**.

CRLs are by default generated and thus downloaded in **DER** format. You can specify *?form=PEM* at the end of the previously given URL to download the CRL in PEM format.

As an example, here are the **CRLDPs** of 2 different CAs that were set up through this guide:

- *https://stream.evertrust.fr/crls/SAGMCA2* will download the CRL for SAGMCA2 through HTTPS in DER format
- *http://stream.evertrust.fr/crls/SAGMCA3?form=PEM* will download the CRL for SAGMCA3 through HTTP in PEM format

2.3.5. External CRL Storages

Creating a Stream External Storage

Stream allows you to push your CRLs into other Stream instances upon generation, but it requires to create an external Stream CRL storage in the product first. This section also assumes that you have already configured `Password` or `Certificate` credentials for the desired stream instance.

To configure an external Stream CRL storage:

1. Log in to the Stream Administration Interface ;

2. Go to **Revocation management** › **External CRL Storage** and click on  ;

3. Fill in the information :

- Select the **Type*** of external CRL storage, Stream for a Stream storage
- The **Name*** to give to that external storage
- The **Description** to add more details about this storage
- Select a list of notifications to send **On execution error** to be alerted if the push to the Stream instance fails
- Add the technical name of the **CA** you wish to push the CRL to. 3 cases can occur:
  - the technical name of the CAs are aligned on both instances: the field should be left blank, as the trigger will by default use the technical name of the CA the CRL is linked to.
  - the technical name of the CAs on the other instance can be deduced from the technical name on the current instance. A template string can be used to format the name correctly.
  - the technical names are not linked in any way. The technical name on the other instance

should be fully spelled out, and a trigger defined for each CA (using duplication)

- Enter the **Endpoint*** of your other Stream instance. This should include the protocol (https://).
- Select a **Credential*** to connect to the Stream instance. Only credentials on the `Stream` target can be selected.
- Choose a **Timeout** for the push request
- Add a **Proxy** to use to connect to the instance, if any

4. Once you've filled all the information, click "Add"

The External CRL Storage is now created and can be used in CA configuration.

Creating an S3 External CRL Storage

Stream allows you to push your CRLs into S3 buckets upon generation, but it implies to configure an external storage first. This section also assumes you have already configured `Password` credentials for a cloud provider if you want to use a cloud storage solution.

To configure an external S3 CRL storage:

1. Log in to the Stream Administration Interface ;

2. Go to **Revocation management** › **External CRL Storage** and click on ;

3. Fill in the information :

- Select the **Type*** of external CRL storage, Amazon S3 for an S3 storage
- The **Name*** to give to that external storage
- The **Description** to add more details about this storage
- Select a list of notifications to send **On execution error** to be alerted if the push to the CRL storage fails
- Add the **Bucket*** of your S3 storage
- Select a **Credential** to connect to the S3 server (AWS format). Only credentials on the `AWS` target can be selected. If no credentials are specified, environment variable values will be used to establish connection.
- Add a **Role ARN** to use when connecting to the S3 provider (only applicable for AWS)
- Select the **Region** to use if the S3 is in the cloud (AWS, GCP)
- Add a **Proxy** to use to connect to the external storage, if any
- If not using an AWS S3 Bucket, add the S3 **Endpoint**
- Choose whether to **Force path style** in URL name
- Reconfigure the **CRL Alias**. By default, the S3 object key will be the technical name of the CA with `.crl` extension. Using template strings, this name can be modified. For example, if the file should be named with an uppercase of the CA's CN with the `.pem` extension, CRL Alias will be `{{ Upper({{ca.signer.dn.cn.1}}) }}.pem`

4. Once you've filled all the information, click "Add"

The External CRL Storage is now created and can be used in CA configuration.

Creating an LDAP External Storage

Stream allows you to push your CRLs into LDAP directories upon generation, but it requires to create an external LDAP storage in the product first. This section also assumes that you have already configured `Password` credentials for the desired LDAP directory.

To configure an external LDAP CRL storage:

1. Log in to the Stream Administration Interface ;

2. Go to **Revocation management › External CRL Storage** and click on  ;

3. Fill in the information :

- Select the **Type*** of external CRL storage, LDAP for an LDAP storage
- The **Name*** to give to that external storage
- The **Description** to add more details about this storage
- Select a list of notifications to send **On execution error** to be alerted if the push to the CRL storage fails
- Add the **Host***, IP or hostname of the LDAP server where the CRL will be pushed into. Don't add "ldap://" or "ldaps://" in the beginning
- Add the **Port*** on which the LDAP server is running (default is 389 for LDAP and 636 for LDAPS)
- Select a **Credential*** to connect to the LDAP server. Only credentials on the `LDAP` target can be selected.
- Add a **Proxy** to use to connect to the external storage, if any
- Enter a **Base DN*** that points the LDAP category to publish the CRL into
- Enter a LDAP search **Filter*** to find the resource where to publish the CRL into. **Example** : *(objectclass=cRLDistributionPoint)*
- Define the **CRL Attribute***, the resource attribute to publish the CRL into
- Choose whether to allow Stream to **follow LDAP referral** URLs
- Choose whether to use the **Secure** LDAPS protocol instead of the regular LDAP protocol
- Choose whether to **Disable hostname validation**, allowing Stream to connect to the LDAP server in LDAPS even if the server certificate does not have the specified hostname as a DNS SAN **(only if Secure is turned on)**

4. Once you've filled all the information, click "Add"

The External CRL Storage is now created and can be used in CA configuration.

Creating an SCP External Storage

Stream allows you to push your CRLs into any server supporting the SCP protocol upon generation. This section also assumes that you have already configured SSH credentials for the desired server.

To configure an external SCP CRL storage:

1. Log in to the Stream Administration Interface ;

2. Go to **Revocation management** › **External CRL Storage** and click on  ;

3. Fill in the information :

- Select the **Type*** of external CRL storage, SCP for an SCP storage
- The **Name*** to give to that external storage
- The **Description** to add more details about this storage
- Select a list of notifications to send **On execution error** to be alerted if the push to the CRL storage fails
- Add the **Host***, IP or hostname of the SCP server where the CRL will be pushed into.
- Add the **Port*** on which the SCP server is running (default is 22 for SSH)
- Select a **Credential*** to connect to the SCP server. Only credentials on the SCP/SFTP target can be selected.
- Choose a **Timeout** for the SCP request
- Choose whether to **Use compression** when pushing the CRL
- Enter a known **Fingerprint** to use mutual authentication. If nothing is specified, no fingerprint check will occur.
- Define the **Path*** where to push the CRL. Using template strings, this path can be dynamically set. For example, if the crl should be pushed to the `crls` root folder with a filename being an uppercase of the CA's CN with the `.pem` extension, path will be `/crls/{{ Upper({{ca.signer.dn.cn.1}}) }}.pem`

4. Once you've filled all the information, click "Add"

The External CRL Storage is now created and can be used in CA configuration.

Creating an SFTP External Storage

Stream allows you to push your CRLs into any server supporting the SFTP protocol upon generation. This section also assumes that you have already configured SSH credentials for the desired server.

To configure an external SFTP CRL storage:

1. Log in to the Stream Administration Interface ;

2. Go to **Revocation management** › **External CRL Storage** and click on ⊕ ;

3. Fill in the information :

- Select the **Type*** of external CRL storage, SFTP for an SFTP storage
- The **Name*** to give to that external storage
- The **Description** to add more details about this storage
- Select a list of notifications to send **On execution error** to be alerted if the push to the CRL storage fails
- Add the **Host***, IP or hostname of the SFTP server where the CRL will be pushed into.
- Add the **Port*** on which the SFTP server is running (default is 22 for SSH)
- Select a **Credential*** to connect to the SFTP server. Only credentials on the `SCP/SFTP` target can be selected.
- Choose a **Timeout** for the SFTP request
- Choose whether to **Use compression** when pushing the CRL
- Enter a known **Fingerprint** to use mutual authentication. If nothing is specified, no fingerprint check will occur.
- Define the **Path*** where to push the CRL. Using template strings, this path can be dynamically set. For example, if the crl should be pushed to the `crls` root folder with a filename being an uppercase of the CA's CN with the `.pem` extension, path will be `/crls/{{ Upper({{ca.signer.dn.cn.1}}) }}.pem`

4. Once you've filled all the information, click "Add"

The External CRL Storage is now created and can be used in CA configuration.

2.3.6. Configuring OCSP

To configure an **OCSP** responder, you first need an OCSP signer.

1. Log in to the Stream Administration Interface.

2. Go to **Revocation Management** › **OCSP Signers** and click on ⊕ at the bottom of the page.

3. Fill in the fields to create an OCSP signer that will sign OCSP requests:

- The **Name** of the OCSP signer: a technical name to identify this signer.
- The **Keystore** where to find the key for this signer.
- The **Key** that this signer will sign with.
- The **DN** of this signer, in X500 format with key=value separated by commas.
- The **Notification on signer expiration** that will notify users via Email or REST.

4. You must then generate the CSR ⤵, sign it using the CA you wish to verify certificate for, and

upload the signed certificate back to Stream



The certificate must be signed with the Key Usage `digitalSignature` (critical) and the Extended Key Usage `OCSPSigning`

5. The OCSP Signer is now uploaded. Additional options are now available:

- The **Response Signing Algorithm**, the hash algorithm that will be used on responses signed by this signer

6. Click the **Save** button at the bottom of the page.

Now your OCSP signer has been configured, OCSP must be enabled on a **Certification Authority**:

7. Go to **Certification Authorities**:

- **Managed CAs**, in the **CRL/OCSP** tab
- **External CAs**, in the **Configuration** tab

8. Toggle the **Enable OCSP** option. New options appear:

- **Compromised CA?** can be toggled if the CA was compromised to make all certificates on this CA act as revoked
- The **Default OCSP signer** to use if no explicit signer is defined in the OCSP request
- The **Archive Cutoff mode** to use on OCSP responses:
  - **Issuer**: the archive cutoff date will be this CA emission date
  - **Retention**: the archive cutoff date will be the OCSP request date plus the retention period

9 Click the **Save** button at the top.

2.4. Managing Certificate Templates & EKUs

2.4.1. Certificate Templates

Stream uses the notion of **Certificate Templates** to add additional verifications when enrolling a certificate.

To define a new certificate template:

1. Log in to the Stream Administration Interface.

2. Go to **Certificates** > **Templates** and click  .

3. In the **General** tab, you can:

- set the template's name
- set the **path length** it will tolerate

- disable the template
- enable proof of possession checking when enrolling with a CSR
- remove the **CA BasicConstraint** (set to false by default)

In the **Duration** part of the tab, you can edit the lifetime of the certificates that will enroll on this template, as well as backdate them should you need to.

In the **Private Key policy** part of the tab, you can choose whether to enforce a usage period for the private key that is detached from the validity of the certificate. Should it be defined, this period must be within the validity period of the certificate. This field is optional in the RFC 5280 but mandatory in the ICAO MRTD 9303 norm (section 7.1.1) and should only be used for signature certificates.

In the **Extra CSR Extensions**, OIDs can be whitelisted to be taken as is from the CSR into the signed certificate.

4. In the **KU & EKU** tab, you can set the **Key Usages** and **Extended Key Usages** of the certificates that will enroll on this template. You can also use your own **EKUs** here. If you want to set up your own **EKUs**, please refer to the *Extended Key Usages* part of this section.

5. In the **Extensions** tab, you can edit the **CRLDPs**, **AIA**, **Authority Information Access**, **Policy**, **Qualified Certificate Statement** of the certificates that will enroll on this template. If you want to, the certificates could use the information of the **CA** they will enroll on, otherwise, you can set specific values in the template. These values will then override those retrieved from the CA.

- If you want to issue Qualified Certificates:
  - ETSI QC Compliance Statement declares that the certificates is a Qualified Certificate.
  - ETSI QC SSCD Statement declares that the private key related to the certified public key resides in a Secure Signature Creation Device.
  - ETSI Retention Period Statement indicates the duration of the retention period of material information.
  - ETSI QC Type Statement indicates which type of document can be signed by the certificate (possible values are: ESEAL, ESIGN, WEB, NONE).
  - ETSI Transaction Limit Statement indicates the limits of the transactions, you must fill every field if enabled.
  - ETSI QC PDS Statement is the PKI Disclosure Statement URI for a specified language.
  - ETSI QC Legislation Statement is an array of country codes.

6. In the **Data Fields** tab, you can enforce your **DNs**, **SANs** and **Extensions** to match certain criteria that can be defined in this section. By default, everything is accepted, meaning that any type and amount of **DNs**, **SANs** and **Extensions** can be used in the certificates and it would successfully enroll on the template.

- If you want to enforce a **Subject DN** policy, then click  in **Subject DN composition**, then select the DN element that you want to put a policy on. You can set this element to be mandatory

or not, to use a default value for that element that can be editable or not, you can also add a whitelist of elements that are accepted values for this DN, or you can instead use a regex to match the DN values that are accepted for this element.

- If you want to enforce a **Subject Alternate Names** policy, you can either click **None** to forbid the use of **SANs** in certificates or you can click **Some** to configure the policy. If you clicked **Some**, click  and select the **SAN element** that you want to enforce a policy upon. You can then input a minimum and maximum number of this **SAN element** to be present in the certificate that will enroll: as an example, if you want to make the use of at least one **DNS SAN** mandatory, use 1 as a minimum number. Finally, you can enforce your **SANs** to match a regex to be considered valid on a certificate.
- If you want to enforce an **Extension** policy, you can either click **None** to forbid the use of **Extensions** in certificates or you can click **Some** to configure the policy. If you clicked **Some**, click  and select the **Extension** that you want to enforce a policy upon. You can then set it mandatory or not, and if supported, give it a default value that can be edited or not.

7. Once you've configured your template, you can click **Save** at the top of the page.



As mentioned previously, if you want your certificates to inherit the **CRLDP**, the **AIA** and the **Policy** from the CA, you must toggle on the **Get from CA** switches and not specify any policy, CRLDP or AIA in the template.

2.4.2. Extended Key Usage

Stream allows you to create and manage your own EKUs as long as you have an OID for it.

To create a custom EKU:

1. Log in to the Stream Administration Interface.

2. Go to **Certificates** > **EKU** then go at the bottom of the page and click .

3. Specify the name you want to give to your custom EKU as well as its OID in the menu, then click "Add".

The EKU should show in the list with the custom switch turned on, as opposed to the standard EKUs that have the custom switch turned off.

2.5. Managing Certificate Lifecycle

Enroll



Stream's RA is not supposed to be a comprehensive registration authority and should only be used when necessary. This simple RA is made for "on the fly" generation only. If you want more advanced RA features to manually enroll certificates, you should consider using Horizon's Web RA.

To enroll a certificate via Stream:

1. Log in to the Stream Administration Interface.

2. Go to **Certificates** > **Enroll**. You'll be prompted to fill the following information:

- **CA** *(select)* : The CA that will issue the certificate. The CA must be managed by Stream;
- **Template** *(select)* : The Stream certificate template to use to issue the certificate;
- **CSR type** : Whether the CSR to sign is in a dedicated file (**File** option) or in the clipboard (**Text** option);
- **CSR** field : The CSR to sign (file or PEM-string).

3. Click the **Enroll** button.

Your certificate should now be visible in the Stream search engine.

Revoke

To revoke a certificate in Stream:

1. Log in to the Stream Administration Interface.

2. Go to **Certificates** > **Search** then find the certificate you want to revoke.

3. Click ⃠ on the certificate you want to revoke. Alternatively, you can click on the certificate's DN then click **Action** > **Revoke**.

Your certificate status should turn red.

Search

To search for certificates in Stream, log in to the Stream Administration Interface and then go to **Certificates** > **Search**.

Here are all the search criteria you can use:

- **CA**: the issuing certificate authority
- **Status**: the validity status of the certificate (valid, revoked or expired)
- **Template**: the certificate template the certificate has been enrolled on
- **Certificate DNs**: information regarding the certificate's DNs
- **Expiration date**: the date when the certificate will expire
- **Issuer**: information regarding the certificate issuer's DNs
- **Serial**: the certificate's serial number

You can combine any number of them to refine your search.

2.6. OpenSSH

2.6.1. Managing Certification Authorities

1. Log in to the Stream Administration Interface.

2. Go to **OpenSSH** > **Certification Authority** from the menu on the left.

3. Input your CA's **internal name**.

4. Select the **Keystore** that contains the key you want to use to generate this CA, then select the key that you want to use. If you do not have a keystore set up yet, please refer to the ***Managing Keystores & Keys*** section.

5. You can also configure KRL generation. To configure this section, please refer to the Key Revocation page. Once you're satisfied with your settings, click "Add".



OpenSSH CAs consist mainly of a keypair used to sign entity certificates and KRL, and do not expire.

2.6.2. Managing Key Revocation

Configuring Key Revocation Lists for a Managed CA

To manage the **KRLs** of a managed CA, you first need to set up a **KRL Policy**:

1. Log in to the Stream Administration Interface.

2. Go to **OpenSSH** > **Certification Authorities** and click on  next to the name of the CA you want to edit the KRL policy of.

3. Define the validity period of your KRL, i.e. the period of time while your KRL is considered valid. The countdown starts at the moment the KRL is generated. If you want your KRLs to be valid for a week, you can type 7 **days**.

4. You can then automate the **KRL generation** using either the **Hard KRL generation**, the **Lazy KRL generation** or both of them in combination:

- The **Hard KRL generation** parameter takes a cron expression in Quartz format and generates the KRL every time that cron expression is valid, without any condition. It is recommended to generate the **KRLs** every day. To generate a new **KRL** every day at 1 A.M., the cron expression is: 0 0 1 * * ?
- The **Lazy KRL generation** parameter takes a cron expression in Quartz format and checks if the KRL needs to be updated, i.e. if a certificate has been revoked, since the last KRL generation. If a certificate has been revoked since the last generation then a new KRL will then be generated, otherwise it will do nothing. It is recommended to have a short time span for the lazy generation so that the KRL always stays up to date. To check for possible KRL updates every 5 minutes, the cron expression is: 0 0/5 * * * ?

5. Click the **Save** button at the top of the page.

Now your KRL policy has been configured, and you've been redirected to the Managed CAs page.

You can then generate manually the CA's first KRL using the ↷ button next to the CA's name that you just configured. If you configured the **Hard** or the **Lazy** generation, your KRL will then automatically be updated according to the cron quartz expression you specified.

Viewing KRLs

1. Log in to the Stream Administration Interface.

2. Go to **OpenSSH** > **KRL**.

3. You can then see information regarding your CAs' KRLs that are going to be detailed below:

- The **CA** column indicates the name of the CA whose KRL is detailed in the line
- The **Number** column indicates the serial number of the KRL. It starts at 1 for the very first KRL generated and is incremented by 1 at each generation. It is displayed in hexadecimal format.
- The **Last update** column indicates the date and time when the current KRL was generated.
- The **Next refresh** column indicates the date and time when the current KRL will be refreshed. It should be equal to the nearest date matching either cron quartz expression you set in the **KRL policy** field (lazy or hard).
- The **download** ⬇ **button** allows you to download your KRL. It also serves as a KRLDP. For more information about KRLDPs in Stream, please refer to next section.
- The **generate** ↷ **button** allows you to manually refresh the KRL and generates a new one.
- The **refresh** ⟳ **button** refreshes the information displayed in the tab, in case a generation happened in between. It **does not** refresh the KRLs, only the displayed information.

Downloading KRLs

Stream allows you to download the **KRLs** of the **CAs** it manages. The standard download URL format is *http(s)://[stream_url]/krls/CA_internal_name*. This URL can be accessed by anyone without prior authentication, either through HTTP or HTTPS.

You need to specify the **Internal name** of the CA to download its **KRL**.

As an example, here us the **KRLDP** of a CA that were set up through this guide:

- *https://stream.evertrust.fr/krls/SAGMCA2* will download the KRL for SAGMCA2 through HTTPS in PEM format

2.6.3. Managing Certificate Templates

Certificate Templates

Stream uses the notion of **Certificate Templates** to add additional verifications when enrolling a

certificate.

To define a new certificate template:

1. Log in to the Stream Administration Interface.

2. Go to **OpenSSH** > **Templates** and click  .

3. In the **General** tab, you can set the template's name, the **type** of SSH certificates it will generate and turn the template on or off. In the **Duration** part of the tab, you can edit the lifetime of the certificates that will enroll on this template, as well as backdate them should you need to. In the **OpenSSH** part of the tab, you can edit the authorized key types as well as the principals required on OpenSSH certificates.

4. Once you've configured your template, you can click **Save** at the top of the page.

2.6.4. Managing Certificate Lifecycle

Enroll

To enroll a certificate via Stream:

1. Log in to the Stream Administration Interface.

2. Go to **OpenSSH** > **Enroll**. You'll be prompted to fill the following information:

- **CA** *(select)* : The CA that will issue the certificate. The CA must be managed by Stream;
- **Template** *(select)* : The Stream certificate template to use to issue the certificate;
- **Public key type** : Whether the Key to sign is in a dedicated file (**File** option) or in the clipboard (**Text** option);
- **Public key** field : The key to sign (file or PEM-string).
- **Principals** field : The principals to sign the certificates for.

3. Click the **Enroll** button.

Your certificate should now be visible in the Stream search engine.

Revoke

To revoke a certificate in Stream:

1. Log in to the Stream Administration Interface.

2. Go to **OpenSSH** > **Search** then find the certificate you want to revoke.

3. Click ⃠ on the certificate you want to revoke. Alternatively, you can click on the certificate's DN then click **Action** > **Revoke**.

Your certificate status should turn red.

Search

To search for certificates in Stream, log in to the Stream Administration Interface and then go to **Certificates** > **Search**.

Here are all the search criteria you can use:

- **CA**: the issuing certificate authority
- **Template**: the certificate template the certificate has been enrolled on
- **Status**: the validity status of the certificate (valid, revoked or expired)
- **Valid after**: the date after which the certificate will be valid
- **Valid before**: the date when the certificate will expire
- **Key ID**: the certificate's key ID

You can combine any number of them to refine your search.

2.7. Managing Keystores & Keys

2.7.1. Keystores in Stream

In Stream, keys are grouped in key containers called **Keystores**.

Stream handles 3 types of Keystores: Software keystores, PKCS#11 HSMs and Cloud KMS. Note that some restrictions apply regarding the supported key types of the HSMs, namely:

- The software keystore supports:
  - RSA key sizes above 1024 bits (the web administration console only offers RSA 2048, RSA 3072, RSA 4096 and RSA 8192);
  - 6 elliptic curves: ECC NIST P-256, ECC NIST P-384, ECC NIST P-521, ECC Brainpool P-256, ECC Brainpool P-384 and ECC Brainpool P-512;
  - 2 Edward curves: ED-448 and ED-25519;
  - 3 MLDSA Algorithms: MLDSA-44, MLDSA-65 and MLDSA-87;
  - 3 MLDSA Algorithms with PreHash: MLDSA-44 + SHA512, MLDSA-65 + SHA512 and MLDSA-87 + SHA512;
- The PKCS#11 keystore crypto capabilities are entirely reliant on the HSM that is used. Generally, RSA keys are all supported, while elliptic curves are not all supported by every HSM vendor. Currently, Edward curves are also not supported by some HSM vendors; PQC is not yet standardized in PKCS#11 so MLDSA support is not yet available.
- Stream can consume the following key types from an AWS KMS instance:
  - RSA 2048, RSA 3072, RSA 4096;
  - ECC NIST P-256, ECC NIST P-384, ECC NIST P-521;

- The AWS KMS currently does not support Edward Curves;
- Stream currently does not support the ECC SECG P-256k1;

- Stream can consume the following key types from an AKV instance:
  - RSA 2048, RSA 3072, RSA 4096;
  - ECC NIST P-256, ECC NIST P-384, ECC NIST P-521;
  - Azure Key Vaults (even the Premium ones) currently do not support Edward Curves;
  - Stream currently does not support the ECC SECG P-256k1;
- Stream can consume the following key types from a GCP CKM instance:
  - RSA 2048, RSA 3072, RSA 4096;
  - ECC NIST P-256 and ECC NIST P-384;
  - The GCP CKM currently does not support Edward Curves.



`EXTERNAL` and `EXTERNAL_VPC` keys on GCP CKM are not supported.

2.7.2. Software keystore

Stream comes installed with a software keystore that can be used to generate RSA and ECDSA keys. To set up a software keystore:

1. Log in to the Stream Administration Interface.

2. Go to **Keystores and keys** and click  .

3. In **Type**, select **Software**. In **Name**, set the name you want to give to your keystore. Optionally, you can add a description to your keystore.

4. Click the **Add** button.

Your keystore should appear in your keystores list with a green circle next to its name.



When using the software keystore, private keys are at some point stored in memory in **plain text**. That represents a huge security flaw since it would just take a memory dump of the Stream machine to be able to recover the private keys.



It is **not recommended** to use the software keystore except for testing or development purposes due to the safety reasons detailed above.

2.7.3. PKCS#11 HSM

Stream supports key management through **PKCS#11 HSMs**.

Stream has been qualified to work with the following **HSMs** but should be working with any **PKCS#11 HSM**:

- Entrust nShield Solo, Entrust nShield Connect, Entrust nShield as a Service
- Atos Proteccio
- Thales Luna (including DPoD), Thales Protect Server
- Utimaco CryptoServer

To set up a PKCS#11 keystore:

1. Log in to the Stream Administration Interface.

2. Go to **Keystores and keys** and click .

3. In **Type**, select **PKCS#11**. In **Name**, set the name you want to give to your keystore. Optionally, you can add a description to your keystore.

4. Input the **full path** of the **PKCS#11 library** (ending in *.so*) of your HSM, then click the parse  button. If your HSM's library was successfully loaded into Stream, you should be seeing your HSM's information. If you get an HSM error, please check the configuration of your HSM. Click "Next".

5. Select the **HSM slot** that you will be using on your HSM for this keystore and input its **PIN code**;

6. Optionally, you can set a Pool Size to your PKCS#11 interface. If disabled, Stream will open a PKCS#11 session every time it needs to sign a certificate, then close it afterwards. If enabled, Stream will open the number of connections specified in the pool size value and maintain them open as long as Stream is running, to be able to directly sign certificates without having to open a PKCS#11 session. This feature comes particularly handy whenever working with a slow HSM, where opening a session is a pretty long operation that can completely ruin performance.

Once you are done, click "Save". Your keystore should appear in your keystores list with a green circle next to its name.

2.7.4. Cloud KMS

Stream supports 3 types of Cloud KMS: Google Cloud Platform (GCP), AWS Key Management Service (KMS) and Microsoft Azure Key Vault (AKV).

Setting up a Google Cloud Key Management (GCP CKM)

1. Log in to the Stream Administration Interface.

2. Go to **Keystores and keys** and click .

3. In **Type**, select **Google Cloud Platform**. In **Name**, set the name you want to give to your keystore. Optionally, you can add a description to your keystore.

4. Select the GCP credential to use to connect to the Cloud Key Management service. If you do not have your GCP CKM credentials set up in Stream yet, please refer to the *Credentials* part of the ***Managing Security*** section.

5. Input the **GCP Project name** in **Project**, the **GCP Server location** to use and the **GCP Key Ring** to use. Additionally, you can specify the **proxy** to use as well as the **timeout period**. Once you are done, click "Add".

Your keystore should appear in your keystores list with a green circle next to its name.

Setting up an AWS Key Management Service (AWS KMS)

1. Log in to the Stream Administration Interface.

2. Go to **Keystores and keys** and click  .

3. In **Type**, select **AWS**. In **Name**, set the name you want to give to your keystore. Optionally, you can add a description to your keystore.

4. Select the AWS credential to use to connect to the AWS Key Management Service. If you do not have your AWS KMS credentials set up in Stream yet, please refer to the *Credentials* part of the ***Managing Security*** section.

5. Input the **AWS server's region** in **AWS Region**. Optionally, you can specify which AWS Role ARN that should be impersonated for that KMS. Additionally, you can specify the **proxy** to use as well as the **timeout period**. Once you are done, click "Add".



To make Stream able to use the keys in the AWS KMS for signature, you need to give it the proper permissions in the AWS console. For more information regarding this topic, please refer to this link, under the "Asymmetric KMS keys for signing and verification".

Your keystore should appear in your keystores list with a green circle next to its name.

Microsoft Azure Key Vault (AKV)

1. Log in to the Stream Administration Interface.

2. Go to **Keystores and keys** and click  .

3. In **Type**, select **Azure Key Vault**. In **Name**, set the name you want to give to your keystore. Optionally, you can add a description to your keystore.

4. Select the AKV credential to use to connect to the Microsoft Azure Key Vault. If you do not have your Microsoft AKV credentials set up in Stream yet, please refer to the *Credentials* part of the ***Managing Security*** section.

5. Specify your Azure vault URL in the **Vault URL** box and the Azure tenant in the **Azure Tenant** box. Additionally, you can specify the **proxy** to use as well as the **timeout period**. Once you are done, click "Add".

Your keystore should appear in your keystores list with a green circle next to its name.

2.7.5. Managing keys in Stream

Regardless of the type of keystores you set up, you can manage the keys through Stream the same way

Adding a key into a keystore

1. Log in to the Stream Administration Interface.

2. Go to **Keystores and keys** and click + on the keystore you want to add the key into.

3. Set the name of the key as well as the key type (RSA, ECDSA or EDDSA) and the key size (for RSA)/key parameter (for ECDSA/EDDSA).

4. For the **Cloud KMSs**, you can set the key to be **Hardware protected** through the dedicated toggle. For the **PKCS#11 HSM**, you can set the key to be **exportable** through the dedicated toggle.

5. Once you set up the key parameters as you want them, click "Add".

The page should refresh and show you the list of keys for the keystore you pushed the key into, where you should see the key you just added.

Viewing the keys of a keystore

1. Log in to the Stream Administration Interface.

2. Go to **Keystores and keys** and click ⌄ on the keystore you want to view.

3. You should see the list of keys on your keystore.

You can then see information about the keys in the keystore:

- The **name** column where you can see the name of the key ;
- The **type** column where you can see the type of algorithm that was used to generate the key. Both RSA and ECDSA are part of the *suiteb* type algorithms ;
- The **key type** column where you can see the algorithm that was used to generate the key as well as the key size/parameter ;
- The **exportable** column indicates if the key is exportable or not.

Deleting a key from a keystore

1. Log in to the Stream Administration Interface.

2. Go to **Keystores and keys** and click ⌄ on the keystore you want to delete the key from.

3. Click the 🗑 icon on the key that you want to delete and click "Confirm" on the prompt.



You cannot delete a key from a keystore if this key is currently used by a CA in Stream. You must first delete the CA that references it and then go over the

deleting procedure.

2.8. Managing Notifications

2.8.1. Email

This section details how to configure the email notifications.

How to create an email notification

1. Log in to Stream Administration Interface.

2. Access emails from the drawer or card: **Notifications** › **Emails**.

3. Click on  .

4. Fill in all mandatory fields in the Notifications details panel.

- The **Name** of the notification.
- The **Lifecycle event** triggering the notification.
  - First, select the entity **Type** can be one of:
    - CA: events on the Certification Authority lifecycle, like expiration of a CA
    - CRL: events on the CRL lifecycle, like a generation or update
    - OCSP Signer: events on the OCSP Signers lifecycle, like expiration of an OCSP signer
    - System: events concerning Stream global objects, like credentials or license expiration
    - Timestamping Signer: events on the Timestamping Signers lifecycle, like expiration of a Timestamping signer
  - Then, select the **Event** you wish to send the notification on.
- Select the notifications to send **On execution error** of this Notification to be alerted if it failed.
- The **Delay before notification sending** is only enabled when the **Event** is about the expiration of an entity. This is the period where the notification will be sent before the expiration date.

5. Fill in all mandatory fields in the Notifications Email details panel.

- The email sender **From** which the mail will be sent.
- The email targets **To** send email.
- The email **Subject**, a template string that will be dynamically evaluated upon email generation.
- The email **Body**, a template string that will be dynamically evaluated upon email generation.
- Set whether the email body **is HTML**. The default value is set to false.



You can click on the ⌄ next to the "Dynamic attributes" section, in order to get a range of possibilities on which dictionaries and functions are available.

6. Click on the save button.

You can edit , duplicate or delete the Email Notification.

2.8.2. REST

This section details how to configure REST notifications.

How to create a REST notification

1. Log in to Horizon Administration Interface.

2. Access REST from the drawer or card: **Notifications** › **REST**.

3. Click on  .

4. Fill in all mandatory fields in the Notifications details panel.

- The **Name** of the notification.
- The **Lifecycle event** triggering the notification.
  - First, select the entity **Type** can be one of:
    - CA: events on the Certification Authority lifecycle, like expiration of a CA
    - CRL: events on the CRL lifecycle, like a generation or update
    - OCSP Signer: events on the OCSP Signers lifecycle, like expiration of an OCSP signer
    - System: events concerning Stream global objects, like credentials or license expiration
    - Timestamping Signer: events on the Timestamping Signers lifecycle, like expiration of a Timestamping signer
  - Then, select the **Event** you wish to send the notification on.
- Select the notifications to send **On execution error** of this Notification to be alerted if it failed.
- The **Delay before notification sending** is only enabled when the **Event** is about the expiration of an entity. This is the period where the notification will be sent before the expiration date.

5. Fill in all mandatory fields in the Notifications Rest details panel.

- The REST **HTTP method and URL** that the HTTP request will use
- The **Proxy** used by Stream to send the HTTP request.
- The **Timeout** to wait before stopping listening to an answer.
- The **Accepted response HTTP code(s)** to consider the request in success state.
- The **Authentication type and credentials** used by the HTTP request, for example, the basic authentication.
- The **Headers** sent along with the HTTP request. Each header value is a template string that will be dynamically evaluated when sending the notification.
- The **Payload** of the HTTP request. It is a template string that will be dynamically evaluated when sending the notification.



You can click on the ⌄ next to the "Dynamic attributes" section, in order to get a

> range of possibilities on which dictionaries and functions are available.

6. Click on the save button.

You can edit , duplicate or delete the Email Notification.

2.9. Managing Security

2.9.1. Authorizations

This section details how to configure the permissions granted to an account, either directly or through a configured role.

Prerequisites

According to the context, you might need to set up:

- [admin-guide:security-roles:::_roles]
- Local accounts

How to add an authorization manually or from a certificate

1. Log in to Stream Administration Interface.

2. Access Authorizations from the drawer or card: **Security** › **Authorizations**.

3. Click on  .

4. Click on  Add Authorization Manually

5. Fill the mandatory fields:

- Either:
  - Fill in an **Identifier***: it can be either a local account identifier or an OpenID Connect identifier (usually email address).
  - Import a certificate by clicking on certificate button  .

6. Click on add button.

How to add an authorization from a search

1. Log in to Stream Administration Interface.

2. Access Authorizations from the drawer or card: **Security** › **Authorizations**.

3. Click on  .

4. Click on  Search and Add Authorization

5. Search by **Identifier** for a local account previously defined.

6. Click on search button.

7. Choose the identifier you want to add.

8. Click on add button.

You can update , see connexion information , or delete Authorization.

How to grant a permission

1. Click on .

Role

2. Select a role previously created (if needed).

Permissions

Stream allows you to manage 2 types of permissions: configuration and lifecycle.

Stream uses wildcard permissions which means you can configure the permissions very thoroughly.

Configuration

For configuration permissions, you can specify:

- the **Section** (ex: Security)
- the concerned **Module** (only for select modules)
- the type of permission: **Audit** (read-only) or **Manage** (read-write, equivalent to *All*).

4. Click on add button.

5. Select a section, then a module, then a submodule if there is, and a right.

6. Click on add button (Don’t forget to save).

7. Click on the save button if you are done.

Lifecycle

For lifecycle permissions, you can specify the concerned **CA** and the concerned **Template** then the type of permission: **Enroll**, **Revoke**, **Search** or **All** of these.

4. Click on add button.

5. Select a module, then a profile, and a right.

6. Click on add button. (don’t forget to save).

7. Click on the save button if you are done.

2.9.2. Credentials

This section details how to configure credentials. Credentials are where credentials for all integrations are regrouped.

How to create credentials

1. Log in to Stream Administration Interface.

2. Access Credentials from the drawer or card: **Security › Credentials**.

3. Click on ⊕.

4. Fill the fields.

- The **Type***: `Certificate` for certificate based authentication, `Login` for login with password credentials, `API Token` for a single value secret (JSON or other) or SSH for SSH Keys secret.
- The **Name*** of the credentials. It should clearly identify it.
- The **Description** to add additional information on these credentials.
- The **Expiration date**. This will be taken from the certificate for `Certificate` credentials, and will be used for notifications on expiration.
- The **Expiration notifications** are [admin-guide:notifications-mail:::_email] or [admin-guide:notifications-rest:::_REST] notifications on event `Credentials expiration` that will run on expiration. Notifications configured here will be sent by the internal monitoring action.
- Certificate:
  - The **PKCS#12*** file containing the authentication certificate and its key.
  - The **PKCS#12 Password*** to open this PKCS#12.
- Credentials:
  - The **Login*** of the account.
  - The **Password*** of the account.
- API Token:
  - The **API Token*** to use.
- SSH:
  - The **SSH identifier***: username to use for SSH connection.
  - The **SSH key*** is the SSH private key in OpenSSH format for SSH connection.

5. Click on the save button.

You can update ✎ or delete 🗑 the Credentials.

2.9.3. Identity Providers

How to configure an Identity Provider

1. Log in to Stream Administration Interface.

2. Access Identity Providers from the drawer or card: **Security › Access Management › Identity Providers**.

3. Click on  .

General tab

4. Select an identity provider type. Currently only OpenID is supported

OpenID connect

5. Fill in all fields:

- The **Name*** will be used to identify this provider on Stream and on the login page.
- **Enabled*** allows to disable the identity provider when access from this authentication source is not needed.
- **Enabled on `UI`*** allows to hide this provider on the login page, but it will still be available via direct API calls.
- The **Provider metadata URL*** is the url where the OIDC provider provides its metadata. For example `https://<oidc server>/.well-known/openid-configuration`.
- The **Client Credentials*** are `Password` credentials containing the client id and secret used to connect to the OIDC provider. They can be created on the go using the `+`.
- The **Scope*** used by Stream during authentication on the identity provider to authorize access to user's details.
- The **Proxy** used to access Provider metadata URL, if any.
- The **Timeout** used for authentication on the identity provider. Must be a valid finite duration. The default value is 10 seconds.
- The **Identifier Claim*** is a template string defining how to construct the identifier from the OpenID Connect claims. For example, if the user identifier is contained in the `login` claim, and should be lower case, then the configured value should be `{{Lower({{login}})}}`.
- The **Name Claim*** is a template string defining how to construct the user name from the OpenID Connect claims. For example, if the user name must be constructed as `family name, GIVEN NAME` and family name is available in the `family_name` claim, given name is available in the `given_name` claim, then the configured value should be `{{family_name}}, {{Upper({{given_name}})}}`

6. Click on the save button.

You can update  or delete  the Identity Provider.



You won't be able to delete an Identity Provider if it is referenced in any other

configuration element.

2.9.4. Local Accounts

How to create local accounts

1. Log in to Stream Administration Interface.

2. Access Local accounts from the drawer or card: **Security** › **Access Management** › **Local Accounts**.

3. Click on  .

4. Fill in the fields:

- The **Identifier***, a meaningful identifier for the account holder. It will be used as a login to access to the solution.
- The **Name*** for the account holder. It will be displayed on the interface when logged in.

5. Click on the create button. The account is created and a password is generated.

How to reset a password on a local account

1. Once a local account is created. Click on  .

You can edit  or delete  a local account. You can reset  a local account password.



You can not delete yourself from local accounts.

2.9.5. Roles

Roles are a way to factor permissions making it easier to configure accounts and track permissions.

Creating a new role

1. Log in to the Stream Administration Interface.

2. Go to **Security** > **Roles** and click  ;

3. Set the **name** of the role you want to create.Optionally, you can add a **description** to the role.

4. Add the **configuration permissions** you want the members of this role to have using the  from **Configuration permissions**. If the role is supposed to have no configuration permission, leave this section empty.

5. Add the **lifecycle permissions** you want the members of this role to have using the  from

Lifecycle permissions. If the role is supposed to have no lifecycle permission, leave this section empty.

Once everything is set up, you can click **Save**.

Managing roles

1. Log in to the Stream Administration Interface.

2. Go to **Security** > **Accounts**. From there, you can see all Stream roles and their associated information.

- The **name** column displays the role's name;
- The **description** column displays the role's description;
- The **permissions** column shows the **straight permissions** that are set up for the role. If the account has any **configuration** permission, it will display and if it has any lifecycle permission it will display ;
- You can **view all the members of a role** using the button;
- You can **delete a role** using the button;
- You can **edit a role**'s information using the button.

2.9.6. Enforce Certificate Authentication

It is possible to enable x509_enforcing parameter in order to authorize only certificate authentication.



This means local accounts will no longer be able to connect on Stream.



When logging in using an X509 certificate, there is no logout option, meaning that the only way to log out is to change the presented certificate in your browser, or to switch to private browsing.

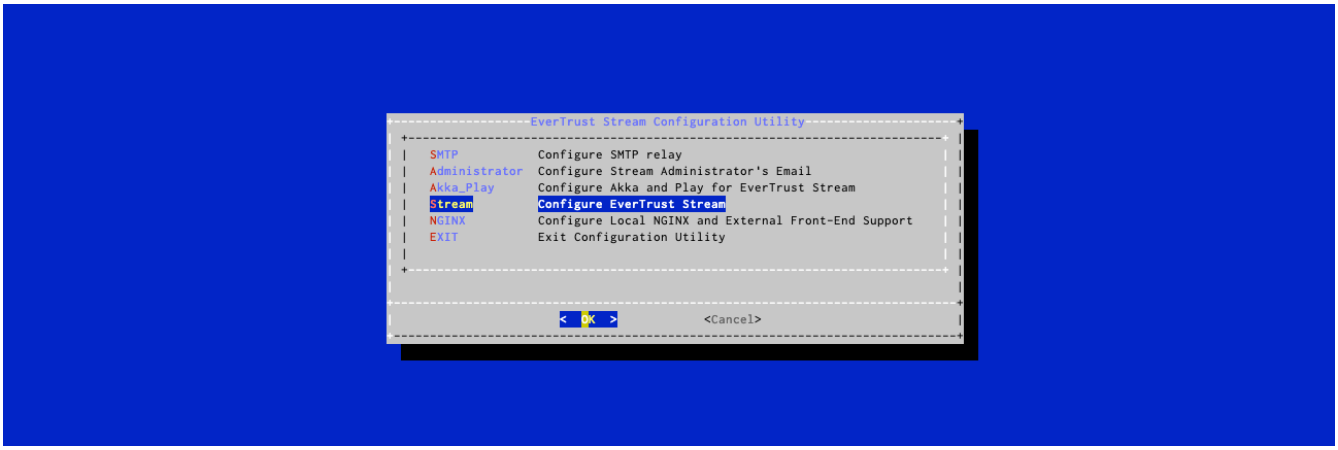
Using Stream configuration utility

Connect to the server with an account with administrative privileges;

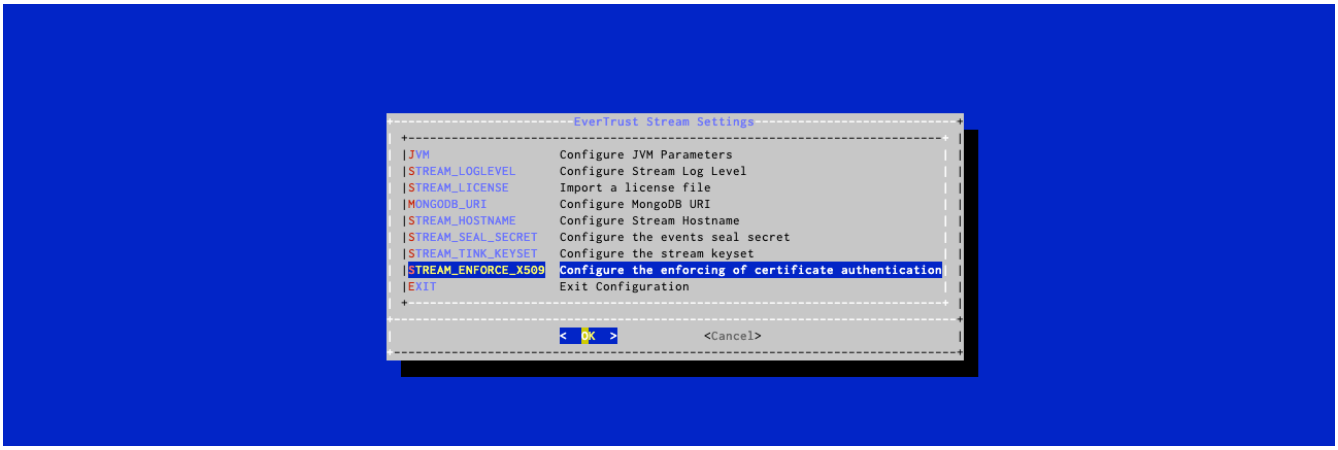
Start the Stream configuration utility by running:

```
# /opt/stream/sbin/stream-config
```

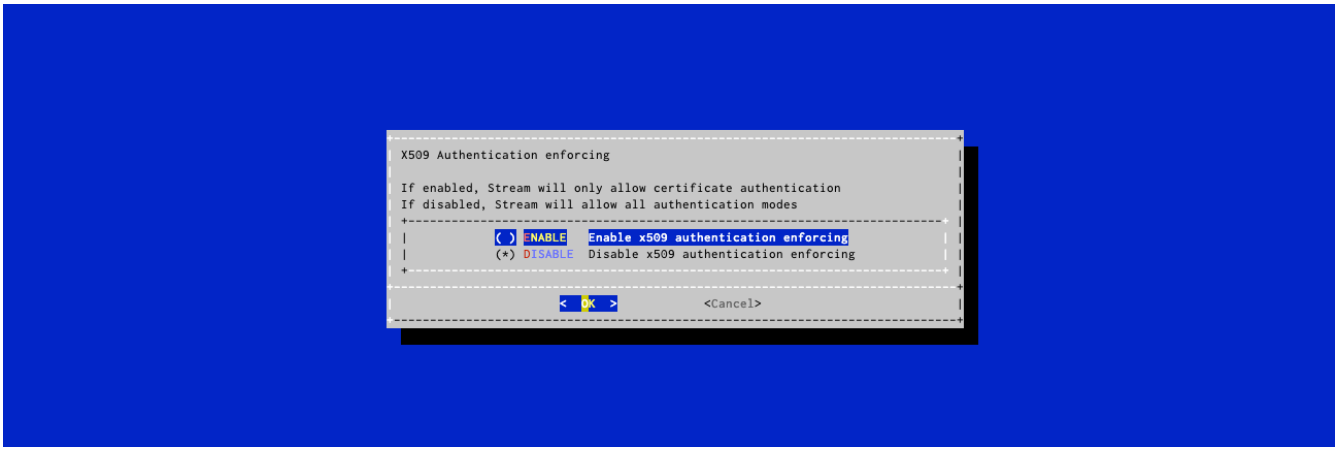
In the main menu, select '**Stream**':



In the Stream menu, select '**STREAM_ENFORCE_X509**':



In the X509 Authentication Enforcing menu, select '**ENABLE**':



For the changes to take effect, you must restart the Stream service by running:

```
# systemctl restart stream
```

X509 Authentication is now enforced.

Re-enable local authentication



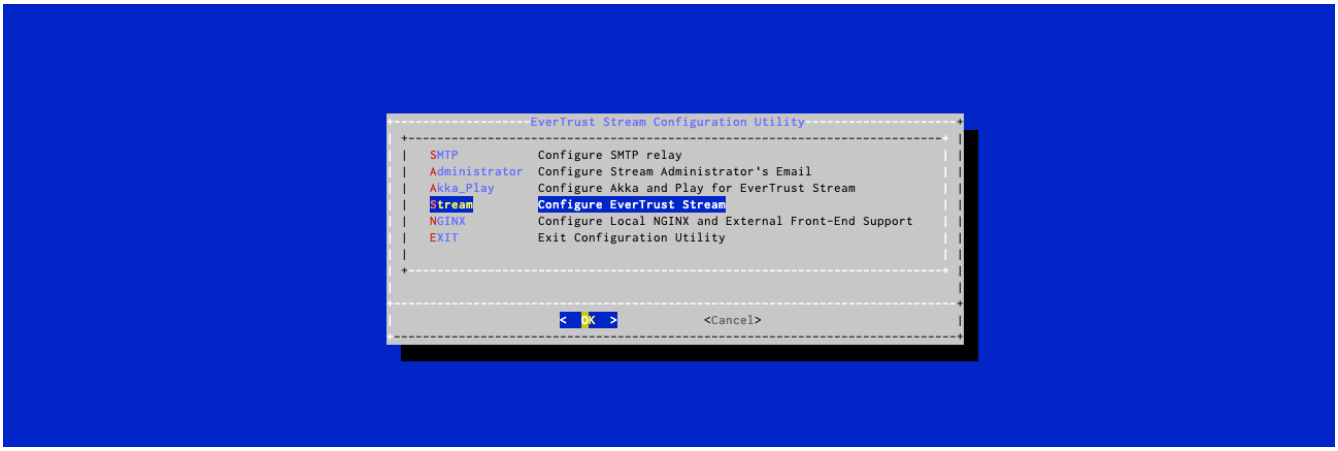
This should be done in a confined and secure environment.

If you lose all available authentication certificates to Stream and want to re-gain access to the administration console, please follow these steps:

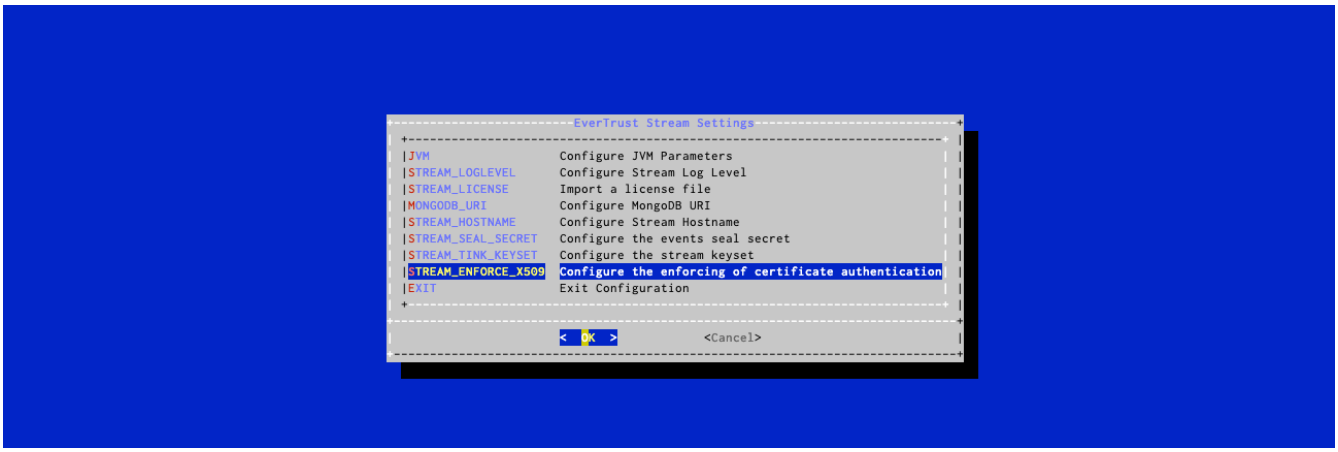
Connect to the server with an account with administrative privileges;

Start the Stream configuration utility by running:

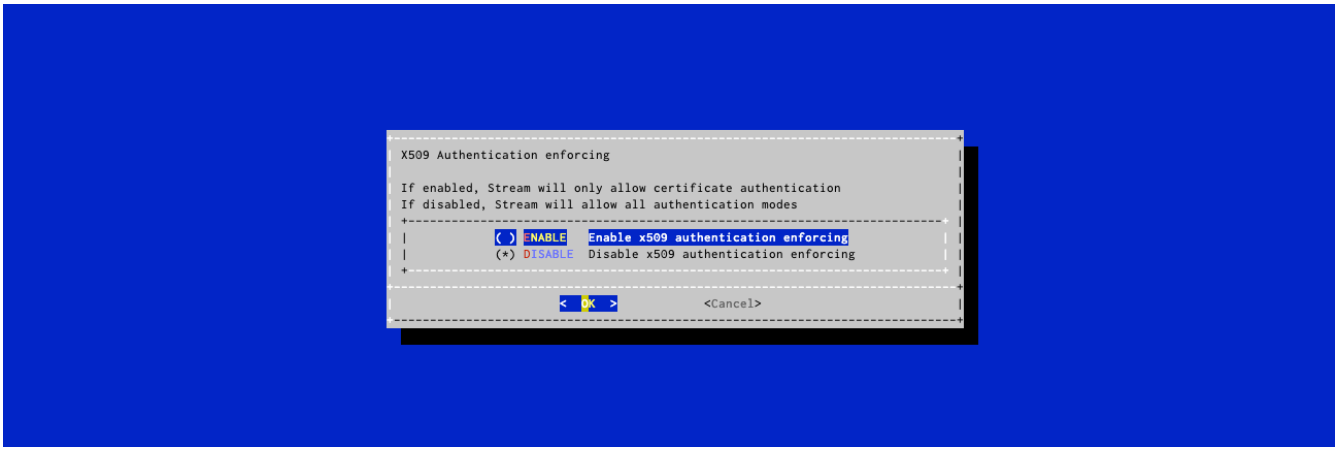
```
# /opt/stream/sbin/stream-config
```



In the Stream menu, select '**STREAM_ENFORCE_X509**':



In the X509 Authentication Enforcing menu, select '**DISABLE**':



For the changes to take effect, you must restart the Stream service by running:

```
# systemctl restart stream
```

Now that the X509 enforcing is disabled, you can log in with the initial administrator account that was created during the bootstrap of the product. If you lost access to that account as well, or if you deleted it, please contact the EVERTRUST support.

2.10. Managing Stream instance

2.10.1. Events

The event system exists to overview the actions happening on Stream.

By default, the events are chained by the following rule: event `n` references event `n-1`. They are signed with the event seal secret set up during the stream installation.

To consult them:

1. Log in to the Stream Administration Interface.

2. Go to **System** > **Events**.

Event integrity reports

To check the integrity of the events, you can run an event integrity report:

1. Log in to the Stream Administration Interface.

2. Go to **System** > **Events Integrity Reports**.

3. Click ⌃;

4. Click `Run`

The integrity of the event chain is checked and can take some time depending on the number of events in the database. Once finished, the report may have different status:

- `Running`: the integrity of the events is currently being checked.
- `Verified`: the event chain is not compromised.
- `Report integrity failure`: the report signature has been compromised.
- `Event integrity failure`: the event chain has been compromised, one event could have been modified or deleted. The event integrity report error provides details about the cause of the integrity failure.



Any compromised object means an account with enough permission to write in the database has been compromised.

Purging/Backup event database



Manual actions regarding the events manipulation should be done with stream turned off and in a confined environment.

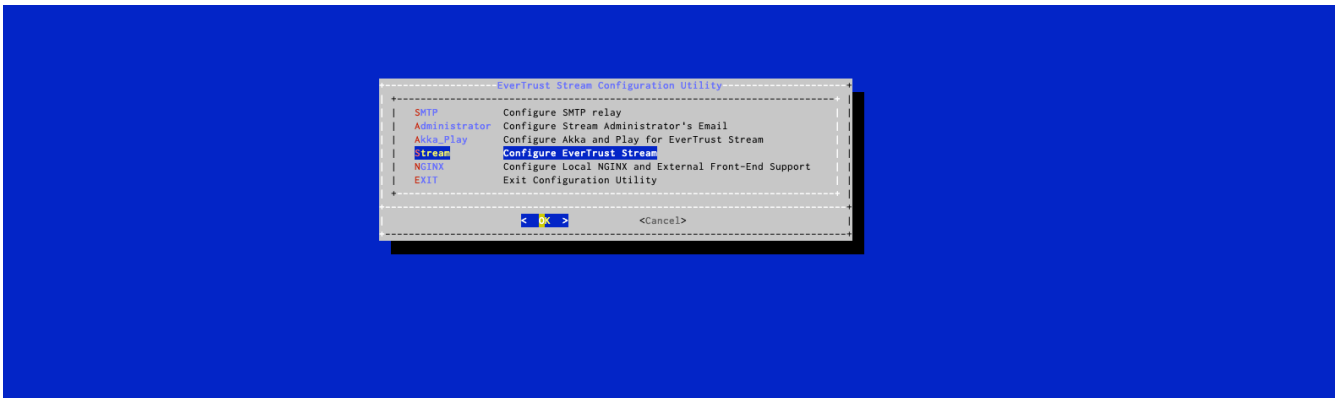
Follow the Backup guide to save your database. Once done, you might want to delete the events in database.

Deletion of events can only be made from the oldest to the newest since events are chained. For example, you might want to delete every event before a date:

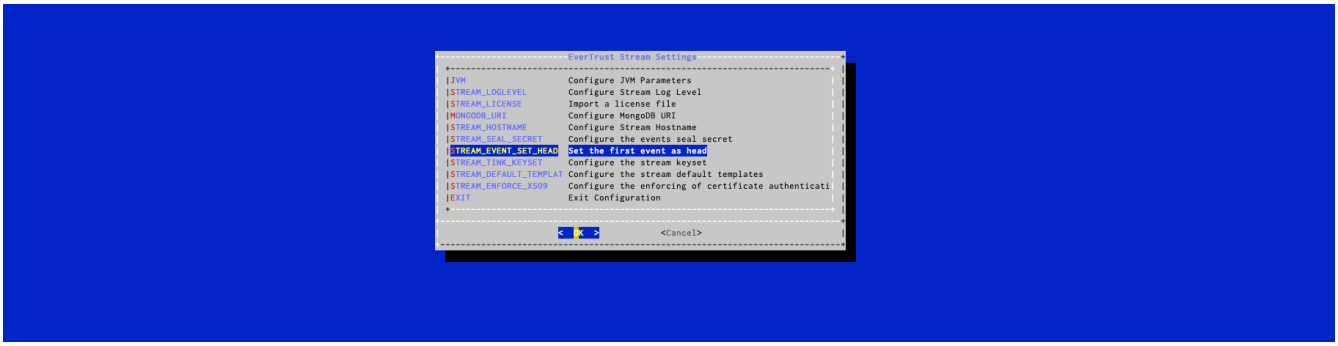
```
use stream;
db.events.deleteMany({"timestamp":{$lt: ISODate("2023-09-20")}});
```

After the deletion of events, the Head is still chained to a deleted event. In order to fix that, you will need to run the `Set the first event as head` in `/opt/stream/sbin/stream-config`:

In the main menu, select '**Stream**':



In the Stream menu, select '**STREAM_EVENT_SET_HEAD**':



Integrity compromised

If an event or event integrity report has been compromised, it means that someone had database access to Stream or one of its backups and manually edited the events to hide specific actions.

You should close all network access to the server and, if necessary, turn off stream. Once confined, you should follow these steps:

1. Follow the Backup guide to back up your database. It may be used to investigate the problem.

2. Analyze the logs (you may use an older verified backup to assess modifications).



Since the database has been compromised, every event should be considered as a non trusted information

3. Based on your assessments, take the appropriate actions. This could mean changing the MongoDB password, changing the server password, revoking stream access certificates or other actions.

4. To resume a normal state, remove every corrupted event following the steps in the event purge guide.

2.10.2. Event codes documentation

All the events displayed in this document work in a similar manner. In case of a failure, the event will display the reason of said failure. This behavior is also valid for warning-status events.

BOOTSTRAP

Bootstrap events relate to the initial setup of the Stream platform.

- **BOOTSTRAP-ADMINISTRATOR-ACCOUNT**
  This event is triggered when installing Stream, it corresponds to the creation of the administrator local identity on Stream.
- **BOOTSTRAP-ADMINISTRATOR-PRINCIPAL**
  This event is triggered when installing Stream, it corresponds to the creation of a link between the administrator account and its rights.
- **BOOTSTRAP-LOCAL-IDENTITY-PROVIDER**
  This event is triggered when installing Stream, it corresponds to the creation of a provider of type Local so that the administrator can connect after startup.
- **BOOTSTRAP-SYSTEM-CONFIGURATION**
  This event is triggered when installing Stream, it corresponds to the creation of internal configuration elements such as the CRON internal monitor.

CA

- **CA-CRL-GEN**
  This event occurs on a CRL Generation request on a CA.
- **CA-CRL-UPLOAD**
  This event occurs when a CRL is being uploaded on a CA.
- **CA-CSR**
  This event occurs when a CSR generation is requested on a CA. This is commonly part of the CA issuing process.
- **CA-ENHANCE**

This event occurs when a legacy CA is being enhanced to a PQC-ready CA.

- **CA-ISSUE**
  This event occurs when a CA is being issued.
- **CA-KRL-GEN**
  This event occurs on a KRL Generation request on a CA.
- **CA-MIGRATE**
  This event occurs when an external CA is being migrated to a managed CA.
- **CA-REVOKE**
  This event occurs on a CA revocation attempt.

CONF

CONF events are triggered when users interact with configuration elements. This includes certificate templates, notification triggers, Certification Authorities…

- **CONF-ADD**
  This event is triggered when a user tries to add a configuration element.
- **CONF-DELETE**
  This event is triggered when a user tries to delete a configuration element.
- **CONF-UPDATE**
  This event occurs when a user tries to modify a configuration element.

CRL

- **CRL-GEN**
  This event occurs on a CRL generation attempt, either requested by application processes or the user.
- **CRL-GET**
  This event occurs on a CRL retrieval attempt from a CRLDP. These are attempted by the application.
- **CRL-SYNC**
  This event is triggered when a failure occurs on a CRL Synchronization.
- **CRL-UPLOAD**
  This event occurs when a user tries to upload a new CRL on a CA.

EVENT COMPLIANCE

- **INVALID-SEAL-PENDING-EVENT**
  This event occurs when a pending event has an invalid seal (indicating data corruption in the pending events collection).
- **UNSEALED-PENDING-EVENT**
  This event occurs when a pending event has no seal (indicating data corruption in the pending events collection).

INTERNAL MONITOR

- **INTERNAL-MONITOR-INIT**
  This event occurs when a bad initialization of the internal monitor happens. It is a failure case, happening for instance when it is not configured
- **INTERNAL-MONITOR-RUN**
  This event occurs when the internal monitor completes successfully.

KRL

- **KRL-GEN**
  This event occurs on a KRL generation attempt, either requested by application processes or the user.
- **KRL-SYNC**
  This event is triggered when a failure occurs on a KRL Synchronization.

LICENSE

- **LICENSE-EXPIRED**
  This event occurs when the license has expired.
- **LICENSE-INVALID**
  This event occurs when the license contains no entitled modules.
- **LICENSE-MODULE-NOT-ENTITLED**
  This event occurs when the requested module is not entitled on the license.

LIFECYCLE

- **LIFECYCLE-ENROLL**
  This event is triggered when an enrollment request for an end-entity certificate is received. The event specifies all the requested certificate fields, as well as CA, keystore and template information. In case of success, the issued certificate PEM is specified. In case of failure, the reason of the failure is specified (e.g.: "Unauthorized DN element").
- **LIFECYCLE-REVOKE**
  This event occurs when a user tries to revoke a certificate. Note that no event is triggered when a certificate expires.

OCSP

- **OCSP-CSR**
  This event is triggered when issuing a CSR for an OCSP Signer.

SECURITY

- **BOOTSTRAP-ADMINISTRATOR**
  This event is triggered when installing Stream, it corresponds to the creation of the initial administrator account (replaced by BOOTSTRAP-ADMINISTRATOR-PRINCIPAL & BOOTSTRAP-

ADMINISTRATOR-ACCOUNT).



Deprecated since version 2.0.0

- **SEC-AUTHENTICATION**
  This event is triggered when a user tries to connect. The identifier (local, OpenID, X509 DN, ...) is specified whether it is a failure or a success.

ACCOUNT

- **SEC-ACCOUNT-ADD**
  This event occurs when an account is created (replaced by authorizations & local accounts).



Deprecated since version 2.0.0

- **SEC-ACCOUNT-DELETE**
  This event occurs when an account is deleted (replaced by authorizations & local accounts).



Deprecated since version 2.0.0

- **SEC-ACCOUNT-UPDATE**
  This event occurs when an account is updated (replaced by authorizations & local accounts).



Deprecated since version 2.0.0

AUTHORIZATION



These events relate to the Security>Access Management>Authorizations tab under configuration.

- **SEC-AUTHORIZATION-ADD**
  This event is triggered when a user tries to create a an authorization object.
- **SEC-AUTHORIZATION-DELETE**
  This event is triggered when a user tries to delete an authorization object.
- **SEC-AUTHORIZATION-UPDATE**
  This event is triggered when a user tries to modify elements inside an authorization object. The event specifies the modified fields.

CREDENTIALS



These events relate to the Security>Credentials tab under configuration.

- **SEC-CREDENTIAL-ADD**
  This event occurs when a user tries creating new credentials.
- **SEC-CREDENTIAL-DELETE**
  This event occurs when a user tries deleting credentials.

- **SEC-CREDENTIAL-UPDATE**
  This event occurs when a user tries updating credentials.

IDENTITY



These events relate to the Security>Access Management>Identity tab under configuration.

- **SEC-IDENTITY-PROVIDER-ADD**
  This event occurs when a user tries creating an identity provider profile.
- **SEC-IDENTITY-PROVIDER-DELEETE**
  This event occurs when a user tries deleting an identity provider profile.
- **SEC-IDENTITY-PROVIDER-UPDATE**
  This event occurs when a user tries modifying an identity provider profile. The modified fields are specified in the event.

LOCAL IDENTITY



These events relate to the Security>Access Management>Local accounts tab under configuration.

- **SEC-LOCAL-IDENTITY-ADD**
  This event is triggered when a user tries creating a local account.
- **SEC-LOCAL-IDENTITY-DELETE**
  This event is triggered when a user tries to delete a local account.
- **SEC-LOCAL-IDENTITY-RESET**
  This event is triggered when executing the reset password workflow.
- **SEC-LOCAL-IDENTITY-UPDATE**
  This event is triggered when a user tries modifying a local account. The modified fields are specified. Updating the password falls in this event.

ROLE



These events relate to the Security>Access Management>Roles tab under configuration.

- **SEC-ROLE-ADD**
  This event is triggered when a user tries to create a new role.
- **SEC-ROLE-DELETE**
  This event is triggered when a user tries to delete a role.
- **SEC-ROLE-UPDATE**
  This event is triggered when a user tries to modify a role. The modified fields are specified in the event.

SERVICE

- **SERVICE-START**
  This event is triggered when the Stream service is started.
- **SERVICE-STOP**
  This event is triggered when the Stream service is manually stopped.

TIMESTAMPING

- **TSA-CSR**
  This event is triggered when issuing a CSR for a Timestamping Signer.

TRIGGER

- **CRL-EXTERNAL-STORAGE**
  This event is triggered when a CRL External Storage runs (replaced by TRIGGER-RUN).

  Deprecated since version 2.0.0

- **TRIGGER-RUN**
  This event occurs when a trigger (External CRL/KRL Storage, Notification) runs.

2.10.3. Proxies

How to configure an HTTP Proxy

1. Log in to Stream Administration Interface.

2. Access HTTP Proxy from the drawer or card: **System** › **HTTP Proxies**.

3. Click on  .

4. Fill the fields:

- The **Name*** of the proxy.
- The **Host*** is the Hostname or IP Address of the proxy.
- The **Port*** of the proxy.

5. Click on the create button to save.

You can update  or delete  the HTTP Proxy.



You won't be able to delete an HTTP Proxy if it is referenced in any other configuration element.

2.10.4. Queue

Queue Configuration

1. Log in to Stream Administration Interface.

2. Access Queues from the drawer or card: **System › Queues**.

3. Click on  .

4. Fill in the fields:

- The **Name*** of the queue. It must be unique.
- The **Description** to add additional information on this queue.
- The **Throttle Duration** and **Throttle Parallelism**. The maximum number of **parallel** request during the **duration**.
- The **Max Size*** of the queue



If the queue is full every new request will be discarded.

- The **Cluster Wide** parameter defines the queue behavior in multi node setup. If not enabled, then the `throttleParallelism` and `throttleDuration` will be the same for all nodes in the cluster. If enabled, then the `throttleParallelism` and `throttleDuration` is generalized for all clusters.

2.10.5. Global configuration

These configurations handle various Stream global parameters directly via the Web Interface.

Internal monitoring

This parameter configures the internal monitoring execution interval. Internal monitoring refers to an action that will check the expiration and usage status of credentials and license, and send the configured notifications if needed.

By default, this action will be executed every day at midnight UTC. The notifications will keep being sent each day for as long as an action is needed.

License configuration

The license configuration panel allows to configure [admin-guide:notifications-mail:::_email] or [admin-guide:notifications-rest:::_REST] notifications to be sent on license expiration: using a notification on the `License Expiration` event and the `Delay before notification sending` field in the notification configuration, notifications configured here will be sent by the internal monitoring action.

2.11. Timestamping

2.11.1. Timestamping Authorities

To configure a Timestamping Authority, a Timestamping Signer and an NTP Client(s) must already be configured.

1. Log in to the Stream Administration Interface.

2. Go to **Timestamping** › **Authorities** and click on ⊕ at the bottom of the page.

3. Fill the fields:

- The unique **Name*** of the Timestamping Authority
- Choose whether to **Enable** it to sign timestamping requests
- Enter the **Policy OID*** this authority manages
- Add a **Timestamping Signer*** that will sign the requests for this authority
- Select the **Accepted Hash Algorithms*** for signature
- Select the **NTP Client(s)*** that will be the time source for the timestamping
- Choose whether to **Check Revocation** of the signer certificate when processing timestamping requests

4. Click "Save" at the bottom.

If everything was ok, the authority now appears in the list.

2.11.2. NTP Clients

1. Log in to the Stream Administration Interface.

2. Go to **Timestamping** › **NTP** and click on ⊕ at the bottom of the page.

3. Fill the fields:

- The unique **Name*** of the NTP client
- Enter a **Description** for additional information about this NTP Client
- Enter the **Host*** where to find the NTP server
- Enter the **Port** where to join the NTP server on. Default is the standard 123 port.
- Select a **Timeout*** for NTP requests
- Choose the NTP parameters like the **Max Stratum**, defining the maximum authorized stratum, the **Max Offset**, defining the maximum offset allowed between local system clock and NTP clock and **Max RTT**, the maximum round trip time allowed.

4. Click "Save" at the bottom.

After these steps, the NTP client now appears in the list.

2.11.3. Timestamping Signers

1. Log in to the Stream Administration Interface.

2. Go to **Timestamping › Signers** and click on ⊕ at the bottom of the page.

3. Fill in the fields to create a Timestamping signer that will sign Timestamping requests:

- The **Name** of the Timestamping signer: a technical name to identify this signer.
- The **Keystore** where to find the key for this signer.
- The **Key** that this signer will sign with.
- The **DN** of this signer, in X500 format with key=value separated by commas.
- The **Notification on signer expiration** that will notify users via Email or REST.

4. You must then generate the CSR ↷, sign it using your Timestamping CA, and upload the signed certificate back to Stream ⇪



The certificate must be signed with the Key Usage `digitalSignature` (critical) and the Extended Key Usage `timeStamping` (critical)

5. The Timestamping Signer is now uploaded. Additional options are now available:

- The **Response Signing Algorithm**, the hash algorithm that will be used on responses signed by this signer

6. Click the **Save** button at the bottom of the page.

2.12. Backup and Restore

This section details how to use the provided EverTrust Tools to back-up and restore Stream if deployed using the RPM package. If you deployed Stream using Docker/Kubernetes, the configuration should be backed-up using the Docker/Kubernetes management platform, and the database should be backed-up using MongoDB tools.

Backup Procedure

This section details how to back up Stream configuration elements (the /opt/stream/etc folder, that includes the Nginx configuration, and the /etc/default/stream configuration file) and the Stream MongoDB database.

The backup tool allows backing up these elements independently.

```
# /opt/stream/sbin/stream-backup --help
```

Stream Backup tool usage: stream-backup [-cdeho:q]
-c | --conf Backup the Stream configuration files
-d | --db Backup the Stream MongoDB Database
-e | --encrypt Encrypt the backup files with the specified passphrase
-h | --help Display the 'stream-backup' help
-o | --output [path] Specify the Stream backup output folder (default: '/opt/stream/var/backup')
-q | --quiet Quiet mode

To back up the configuration files, run the following command:

```
# /opt/stream/sbin/stream-backup -c
```

The configuration files backup consists of a compressed archive (`.tar.gz`) located under `/opt/stream/var/backup/`.

To back up the MongoDB database, run the following command:

```
# /opt/stream/sbin/stream-backup -d
```

The MongoDB database backup consists of a compressed file (`.gz`) located under `/opt/stream/var/backup/`.

To run a complete backup, execute the following command:

```
# /opt/stream/sbin/stream-backup -c -d
```



- The backup output folder can be overridden using the `-o | --output` parameter
- The backup tool can operate in quiet mode (when scheduled in a cron job) using the `-q | --quiet` parameter
- If you want to encrypt your back-up files, use the `-e | --encrypt` parameter. The backup tool will prompt you for a passphrase. The back-up will be encrypted using AES-256.

Restoration Procedure

This section details how to restore a Stream back-up that was generated using the `stream-backup` tool. The restoration happens using the `stream-restore` tool.

```
# /opt/stream/sbin/stream-restore --help
```

Stream restore tool usage: stream-restore
-a | --archive [filepath] The encrypted backup file to restore
-c | --conf [filepath] The path to the Stream configuration backup file
-d | --db [filepath] The path to the Stream database backup file
-m | --mongo_uri [MongoDB URI] The MongoDB URI to back-up the database into (optional)
-h | --help Displays the 'stream-restore' help
-q | --quiet Quiet mode

Whenever trying to restore a backup, you need to stop the Stream service first:

```
# systemctl stop stream
```

To restore an unencrypted configuration backup, run the following command:

```
# /opt/stream/sbin/stream-restore -c [configuration backup archive path]
```

To restore an unencrypted MongoDB database backup, run the following command:

```
# /opt/stream/sbin/stream-restore -d [MongoDB backup archive path] -m [MongoDB URI]
```

The MongoDB URI is optional: if not provided, the script will try to infer it from the /etc/default/stream file. If it cannot be inferred and none is provided, the restore will fail.

To restore an encrypted backup archive, run the following command:

```
# /opt/stream/sbin/stream-restore -a [encrypted backup archive path] -m [MongoDB URI]
```

The restoration tool will prompt you for the passphrase that was used to encrypt the backup. If the archive contains only a configuration backup, the script will perform the equivalent of the `-c` parameter. If the archive contains only a database backup, the script will perform the equivalent of the `-d` parameter, and you might need to provide the MongoURI through the `-m` parameter. If the archive contains both a database and a configuration backup, both of them will be restored.

When the restoration is complete, you can start Stream again using the following command:

```
# systemctl start stream
```

2.13. Dictionaries

Here is the list of available dictionary keys to use in computation rules and template strings, depending on the usage.

Certificate Authority

This dictionary regroups the information of a Certificate Authority.

Key	Description	Type
ca.name	The technical name of the ca	Single value
ca.type	The type of ca (managed or external)	Single value
ca.signer	The values from the signer	Signer dictionary

OCSP Signer

This dictionary regroups the information of an OCSP signer.

Key	Description	Type
ocsp.name	The technical name of the ocsp signer	Single value
ocsp.signer	The values from the signer	Signer dictionary

Timestamping Authority

This dictionary regroups the information of a Timestamping authority.

Key	Description	Type
tsa.name	The technical name of the timestamping signer	Single value
tsa.signer	The values from the signer	Signer dictionary

CRL

This dictionary regroups the information of a CRL.

Key	Description	Type
crl.ca.name	The technical name of the ca that signed the CRL	Single value
crl.ca.type	The type of the ca that signed the CRL	Single value

Key	Description	Type
crl.number	The CRL number	Single value
crl.this_update	The value of `this_update`	Single value
crl.next_update	The value of `next_update`	Single value
crl.next_refresh	The value of `next_refresh`	Single value
crl.size	The number of certificates in the crl	Single value
crl.eidas	"true" if the CRL is eidas compliant, else "false"	Single value
crl.error	The value of the error if the CRL generation failed	Single value

Credentials

This dictionary regroups the information of a Credential.

Key	Description	Type
credentials.name	The credentials name	Single value
credentials.description	The credentials description	Single value
credentials.expires	The credentials expiration date	Single value
credentials.type	The credentials type	Single value
credentials.target	The credentials target	Single value

In a rest notification, headers can be enriched using the credentials values:

Key	Description	Type
credentials.login	The credentials login value for Password Credentials	Single value
credentials.password	The credentials password value for Password Credentials	Single value
credentials.secret	The credentials secret value for Raw Credentials	Single value

License

This dictionary regroups the information of the Stream license.

Key	Description	Type
license.expires	The license expiration date	Single value
license.modules	The enabled modules	Single value

Sub dictionaries

These dictionary cannot be used alone but can be completed with one of the other ones. For example, a valid key is:

```
ocsp.signer.dn.cn.1
```

Signer dictionary

Key	Description	Type
dn	The full dn of the certificate in TYPE=value form	Single valued
dn.<dn field type>	All values of subject field of type `dn field type`	Multi valued
dn.<dn field type>.<index>	Value of subject field of type `dn field type` at index `index`	Single value
sans.<sans field type>	All values of subject field of type `sans field type`	Multi valued
sans.<sans field type>.<index>	Value of subject field of type `sans field type` at index `index`	Single value
issuer	The full dn of the issuer of the certificate in TYPE=value form	Single valued
not_before	Value of the start date of the certificate	Single value
not_after	Value of the expiration date of the certificate	Single value
serial	The certificate serial	Single valued
thumbprint	The certificate thumbprint	Single valued
public_key_thumbprint	The certificate public key thumbprint	Single valued
key_type	The certificate key type	Single valued
signing_algorithm	The certificate signing algorithm	Single valued
pem	The PEM encoded certificate	Single valued



The valid dn field types are: cn, uid, serialnumber, surname, givenname, unstructuredaddress, unstructuredname, e, ou, organizationidentifier, uniqueidentifier, street, st, l, o, c, description, dc.



The valid SAN field types are: `rfc822name`, `dnsname`, `uri`, `ipaddress`, `othername_upn`,

`othername_guid`, `registered_id`.



All indexes start at 1

2.14. Computation rule

Computation Rules are expressions that describe operations to apply to dictionary keys. These keys can come from diverse data sources such as a certification request or a user entry. The available operations and their usage are detailed in this part.

Example

Let's start by an example:

My CSR contains a DNSNAME subject alternate name with the following value:

```
host.evertrust.fr
```

I want my final certificate to have 2 SANs, this value and its short name: "host".

In order to do that, in **Profile › Certificate Template › Subject Alternate Names** , I add a DNSNAME SAN with the following computation rule:

```
[{{csr.san.dnsname.1}}, Extract({{csr.san.dnsname.1}}, "(.*?)\.", 1)]
```

This will output, in my final certificate, two SANs with values:

```
host.evertrust.fr, host
```

To explain this result, the value "host.evertrust.fr" was retrieved by choosing the first DNSNAME SAN of the CSR: `{{csr.san.dnsname.1}}`. The function `Extract` extracted the first catching group from the regex `(.*?)\.`, resulting in the "host" value.

The computation rule language has a lot more possible operations, allowing complex use cases to become reality.

Dictionary keys

Dictionary keys are a way to name the information from the available sources. For instance, for a webra enroll, the available sources are the given csr, the webra enroll form data and the principal information if it is authenticated. The full list of available dictionary keys is available on the dictionary page.

Enrollment

A key can reference a single element or a list of elements. It is separated in three main parts: the source of data (csr, webra enroll data form), the section of the data, and an optional number

For example, the following is a valid key with these 3 parts:

`{{csr.subject.cn.1}}`

The csr is the data source, the subject.cn the requested information and the 1 is the index. It allows to retrieve the first, common name from the subject, from the CSR.

Without an index, the key is still valid, but it will output all the corresponding values. For example

`[[csr.subject.ou]]`

This retrieves all the ou from the subject, from the CSR.



When a key is expected to output a single value it should be written as a single dictionary key, and one outputting a list of values as a multi dictionary key, otherwise it will be none.

Basic expressions

Basic string expressions
The following expressions are evaluated as a string or None.

Expression Name	Syntax	Allowed Values	Description	Example
Single dictionary key	{{<key>}}	key: a-zA-A-._	This retrieves a key value from the dictionary, none if it does not exist	{{csr.subject.cn.1}}
Number	<number>	number: -\d+	This will output the given number	-4
Literal	"<literal>"	literal: any string	This will output the given literal	"iAmAString"
Null	NULL	NULL	This will output None	NULL
Now	NOW	NOW	This will output the current instant	NOW

Basic list expressions
The following expressions are evaluated as a list of string or None.

Expression Name	Syntax	Allowed Values	Description	Example
Multi dictionary key	[[<key>]]	key: a-zA-A-._	This retrieves all values that start with key from the dictionary	[[admin-guide:computation_rules:::csr.subject.cn]]
Array	[<simpleExpression>, ... <simpleExpression>]	simpleExpression: any expression that will be evaluated to a single element	This will output a multi expression composed of all inserted simple expressions	["iAmAString", {{csr.san.dnsname.1}}]

Quick reference



Function names are not case sensitive but keys are

Function Name	Syntax
Upper	Upper(`expression`:<expression>)
Lower	Lower(`expression`:<expression>)
Trim	Trim(`expression`: <expression>)
Substr	Substr(`expression`: <expression>, `start`: <number>)
Substr	Substr(`expression`: <expression>, `start`: <number>, `end`: <number>)
Concat	Concat(`expression`: <expression>, ... <expression>)
Extract	Extract(`expression`: <expression>, `regex`: <literal>)
Extract	Extract(`expression`: <expression>, `regex`: <literal>, `group`: <number>)
Replace	Replace(`expression`: <expression>, `regex`: <literal>, `replacement`: <expression>)
OrElse	OrElse(`expression`: <expression>, ... <expression>)
Match	Match(`expression`: <simpleExpression>, `regex`: <literal>)
DateTimeFormat	DateTimeFormat(`expression`: <simpleExpression>, `format`: <literal>)
Get	Get(`expression`: <multiExpression>, `index`: <number>)
First	First(`expression`: <multiExpression>)
Last	Last(`expression`: <multiExpression>)

Function Name	Syntax
Filter	Filter(expression: <multiExpression>, regex: <literal>)
Slice	Slice(expression: <multiExpression>, start: <number>)
Slice	Slice(expression: <multiExpression>, start: <number>, end: <number>)

Any expression functions

Upper

```
Upper(expression:<expression>)
```

This outputs the result evaluated from `expression` with only upper case characters and None if no value was evaluated

```
Upper("string") => "STRING"
Upper(["string1", "string2"]) => ["STRING1", "STRING2"]
```

Lower

```
Lower(expression:<expression>)
```

This outputs the result evaluated from `expression` with only lower case characters and None if no value was evaluated

```
Lower("STRING") => "string"
Lower(["STRING1", "STRING2"]) => ["string1", "string2"]
```

Trim

```
Trim(expression:<expression>)
```

This outputs the trimmed result evaluated from `expression` and None if no value was evaluated

```
Trim(" STRING") => "STRING"
Trim(["string1 ", "  string2  "]) => ["string1", "string2"]
```

Substr

```
Substr(expression: <expression>, start: <number>)
```

This outputs the substring from index `start` to the end of the string evaluated from `expression` and None if no value was evaluated or the result of substring is empty. `start` can be negative and it will be computed from end of string.

```
Substr("STRING", 2) => "TRING"
Substr(["string", "longerString", "s"], -2) => ["ng", "ng", "s"]
Substr("tooShort", 15) => None
```

Substr

```
Substr(expression: <expression>, start: <number>, end: <number>)
```

This outputs the substring from index `start` to `end` of the string evaluated from `expression` and None if no value was evaluated or the result of substring is empty. `start` and `end` can be negative and it will be computed from end of string.

```
Substr("STRING", 2, 4)  => "TRI"
Substr(["string", "longerString", "s"], 2, -2) => ["tri", "ongerStri"]
Substr("tooShort", -2, 4) => None
```

Concat

```
Concat(expression: <expression>, ...<expression>)
```

This outputs the concatenation of evaluated expressions: if they are all simple expression, a string concatenation will take place, otherwise an array with all the values will be evaluated. If the final result is empty, None will be returned.

```
Concat("start", " middle ", "end") => "start middle end"
Concat(["string1", "string2", "string3"], "string4") => ["string1", "string2",
"string3", "string4"]
```

Extract

```
Extract(expression: <expression>, regex: <literal>)
```

This extracts from the evaluated `expression` string(s) the part that matches the `regex`

```
Extract("abcd@domain.com", ".*@") => "abcd@"
Extract(["string1", "string2", "string3"], "\d") => ["1", "2", "3"]
```

Extract

```
Extract(expression: <expression>, regex: <literal>, group: <number>)
```

This extracts from the evaluated `expression` string(s) the group at index `group` that matches the `regex`

```
Extract("abcd@domain.com", "(.*)@", 1) => "abcd"
Extract(["string1", "string2", "string3"], "(.*)\d", 1) => ["string", "string",
"string"]
```

Replace

```
Replace(expression: <expression>, regex: <literal>, replacement: <expression>)
```

This replaces parts of the evaluated `expression` string(s) that matches the `regex` with the evaluated `replacement`. If `replacement` is None, values will be replaced by an empty string.

```
Replace("abcdATdomain.com", "AT", "@") => "abcd@domain.com"
Replace(["string1", "string2", "string3"], "\d", CONCAT("This", " was ", " a number"))
=> ["stringThis was a number", "stringThis was a number", "stringThis was a number"]
```

OrElse

```
OrElse(expression: <expression>, ...<expression>)
```

This outputs the first non None result of the given expressions, or None if they are all None

```
OrElse({{not.a.value}}, "abcd@domain.com") => "abcd@domain.com"
OrElse([[no.values]], "value") => ["value"]
OrElse([[no.values]], {{not.a.value}}) => None
```

String functions



The following functions output a string or None.

Match

```
Match(expression: <simpleExpression>, regex: <literal>)
```

This outputs the expression if it matches the regex, otherwise None

```
Match("abcd", "[a-z]+") => "abcd"
Match("abcd", "\d+") => None
```

DateTimeFormat

```
DateTimeFormat(expression: <simpleExpression>, format: <literal>)
```

This outputs the expression formatted as format. If expression is not a date, no formatting takes place. Available formats are:

- Custom format in Java DateFormatter syntax
- MILLIS
- BASIC_ISO_DATE
- ISO_LOCAL_DATE
- ISO_OFFSET_DATE
- ISO_DATE
- ISO_LOCAL_TIME
- ISO_OFFSET_TIME
- ISO_TIME
- ISO_LOCAL_DATE_TIME
- ISO_ZONED_DATE_TIME
- ISO_DATE_TIME
- ISO_ORDINAL_DATE
- ISO_WEEK_DATE
- ISO_INSTANT
- RFC_1123_DATE_TIME

```
DateTimeFormat(NOW, "MILLIS") => "1709290260764"
DateTimeFormat(NOW, "hh:mm:ss") => "10:54:57"
```

Get

```
Get(expression: <multiExpression>, index: <number>)
```

This outputs the string at `index` index in the `expression` list, and None if the index does not exist. The index can be negative to get from the end of the list.

```
Get(["string1", "string2", "string3", "string4"], -2) => "string3"
Get(["string1", "string2"], 3) => None
```

First

```
First(expression: <multiExpression>)
```

This outputs the first string of the `expression` list, and None if it does not exist. The index can be negative to get from the end of the list.

```
First(["string1", "string2", "string3", "string4"]) => "string1"
First([[no.values]]) => None
```

Last

```
Last(expression: <multiExpression>)
```

This outputs the last string of the `expression` list, and None if it does not exist. The index can be negative to get from the end of the list.

```
Last(["string1", "string2", "string3", "string4"]) => "string4"
Last([[no.values]]) => None
```

List of string functions



The following functions output a list of string or None.

Filter

```
Filter(expression: <multiExpression>, regex: <literal>)
```

This outputs a list of string from `expression` that matches the `regex`, None if none matches

```
Filter(["string1", "string2", "match"], "[a-z]+") => ["match"]
Filter(["string1", "string2"], "[a-z]+") => None
```

Slice

```
Slice(expression: <multiExpression>, start: <number>)
```

This outputs the slice of the `expression` list between `start` index and its end, or None if the slice is invalid. The index can be negative to get from the end of the list.

```
Slice(["string1", "string2", "string3", "string4"], -2) => ["string3", "string4"]
Slice(["string1", "string2"], 3) => None
```

Slice

```
Slice(expression: <multiExpression>, start: <number>, end: <number>)
```

This outputs the slice of the `expression` list between `start` and `end` index, or None if the slice is invalid. The index can be negative to get from the end of the list.

```
Slice(["string1", "string2", "string3", "string4"], 1, 3) => ["string1", "string2",
"string3"]
Slice(["string1", "string2"], 3) => None
```

2.15. Template Strings

Template Strings are augmented strings. They can be used as normal text but can also be augmented:

Using dictionary values

Using the following format, a dictionary key will be interpreted to its value when sending the notification:

```
{{<dictionary key>}}
```

Example:

```
I am enrolling on {{ca.name}}
```

Depending on the notification event, values will be added to context to be interpreted.



If the value is not available in the context, the dictionary value will not be replaced

Using computation rules

Using the following format, a computation rule will be interpreted to its value when sending the notification:

```
{{<computation rule>}}
```

Example:

```
I am enrolling on {{ Lower({{ca.name}}) }}
```

Depending on the notification event, values will be added to context to be interpreted in the computation rule.



If the computation rule result is None, an empty string will be displayed. If it is an array, it will be in a comma separated string