EVERTRUST

EverTrust Stream documentation v1.1

Installation Guide

EVERTRUST

Table of Contents

1. Introduction

1.1. Description

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

1.2. Prerequisites

1.2.1. Choose an installation method

We offer two installation modes:

- A package-based installation on a server running **CentOS/RHEL 7.x/8.x 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.

1.2.2. 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.

2. Installing on CentOS/RHEL

2.1. Pre-requisites

This section describes the system and software pre-requisites to install Stream.

2.1.1. System pre-requisites

The following elements are considered as system pre-requisites:

- A server running EL [7.x-8.x] x64 (CentOS / RHEL) with the network configured and **SELinux** disabled;
- Base and EPEL CentOS / RHEL [7.x-8.x] x64 repositories activated;
- An access with administrative privileges (root) to the server mentioned above;

2.1.2. Software pre-requisites

The following elements are considered as software pre-requisites:

- The Stream installation package: '***stream-1.1-1.noarch.rpm***';
- The MongoDB Community Edition package available from the MongoDB web site;
- EPEL repository activated.

NOTE As a reminder, EPEL can be activated on CentOS / RHEL by doing the following:

```
yum install epel-release
```

2.2. Installation

2.2.1. Installing MongoDB

NOTE Mongo DB version 4.2.x to 5.x.x are supported by Stream

Download the last version of the following Mongo DB 5.x RPMs from the MongoDB web site:

- `mongodb-org`
- `mongodb-org-mongos`
- `mongodb-org-server`
- `mongodb-org-shell`
- `mongodb-org-tools`

Download the last version of the Mongosh RPM from the mongosh github

- mongodb-mongosh

Upload the downloaded RPMs through SCP on the server under `/root`;

Using an account with privileges, install the RPMs using 'yum'. For example, to install MongoDB version 5.0.1, run the following command from the folder containing the RPMs:

```
$ yum install mongodb-org*
$ yum install mongodb-mongosh
```

Enable the service at startup with the following command:

```
$ systemctl enable mongod
```

Start the `mongod` service with the following command:

```
$ systemctl start mongod
```

Verify that you can connect to the Mongo instance by running the mongo shell:

```
$ mongo
```

NOTE You can disconnect from the shell with ^D

2.2.2. Installing NGINX

1. Access the server through SSH with an account with administrative privileges;
2. Install the NGINX web server using the following command:

   ```
   $ yum 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
   ```

2.2.3. Installing Stream

Installation from the EverTrust repository

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=0
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.

Installing from RPM

Upload the file '***stream-1.1-1.noarch.rpm***' through SCP under `/root`;

Access the server through SSH with an account with administrative privileges;

Install the Stream package with the following command:

```
$ yum localinstall /root/stream-1.1-1.noarch.rpm
```

NOTE

Installing the Stream package will install the following dependencies:

- `dialog`
- `java-11-openjdk-headless`

Please note that these packages may have their own dependencies.

2.2.4. Configuring the Firewall

Access the server through SSH 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
```

2.3. Configuration

2.3.1. Initial Configuration

Generating a Tink keyset

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

- A plaintext keyset (not protected);
- A GCP keyset (protected by a master key in a GCP KMS);
- An AWS keyset (protected by a master key in an AWS KMS).

Stream comes with '**tinkey**' client to manage the generation of a tink keyset.

Here is how to generate a tink keyset:

Generating a plaintext keyset

```
$ /opt/stream/sbin/tinkey generate-keyset --out=/opt/stream/etc/stream.keyset
```

Generating a GCP protected keyset

```
$ /opt/stream/sbin/tinkey generate-keyset --out=/opt/stream/etc/stream.keyset --master
-key-uri=gcp-kms://<GCP master key path>
```

Generating an AWS protected keyset

```
$ /opt/stream/sbin/tinkey generate-keyset --out=/opt/stream/etc/stream.keyset --master
```

```
-key-uri=aws-kms://<AWS master key path>
```

Once the keyset is generated, the following commands need to be run:

```
$ chown stream:stream /opt/stream/etc/stream.keyset
```

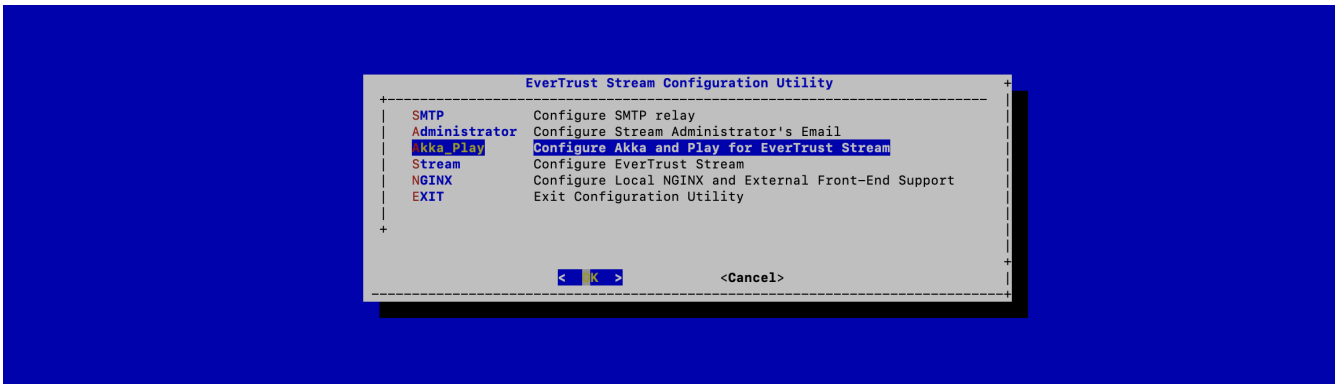
Generating a Play secret

Access the server through SSH with an account with administrative privileges;

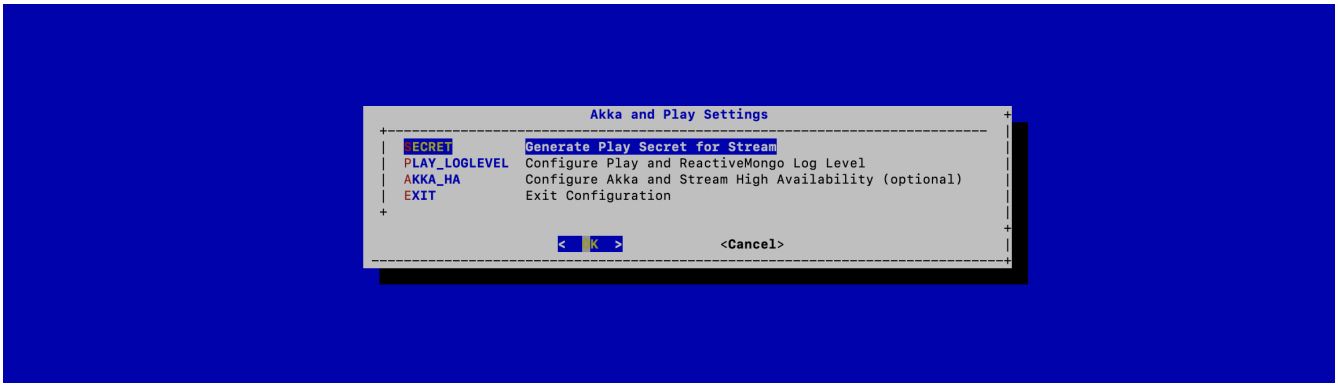
Start the Stream configuration utility by running:

```
$ /opt/stream/sbin/stream-config
```

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

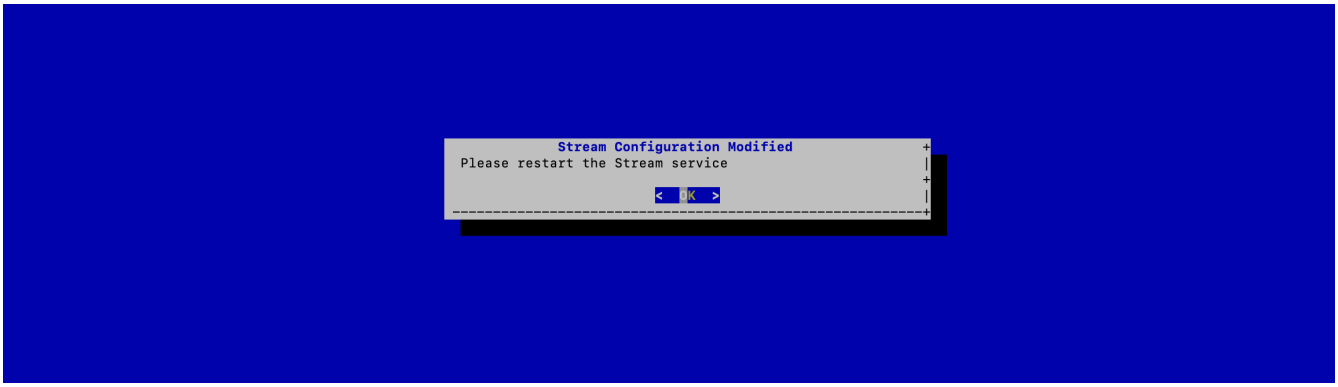


In the Akka_Play menu, select '**SECRET**':



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

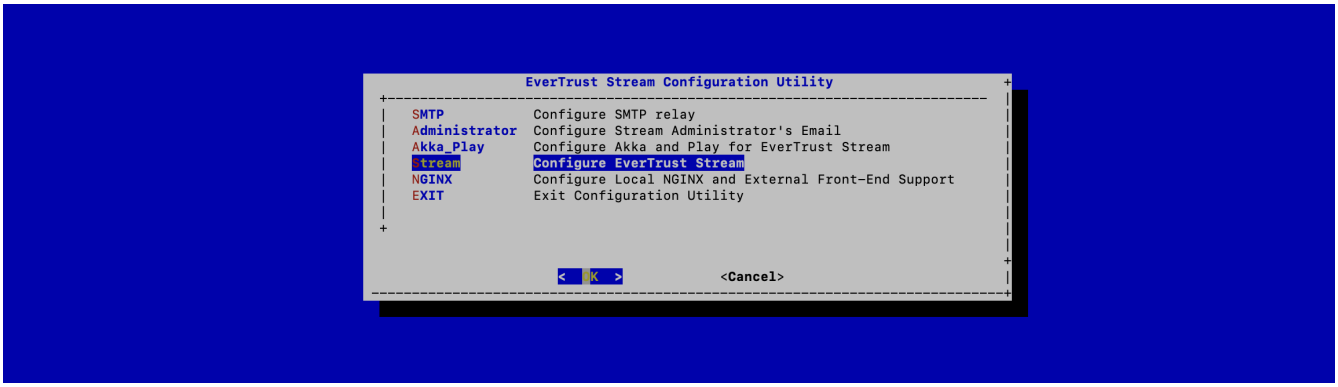
Stream allows you to configure the ***Xms*** (minimum memory allocation pool) and ***Xmx*** (maximum memory allocation pool) parameters of the JVM running Stream using the configuration tool.

Access the server through SSH with an account with administrative privileges;

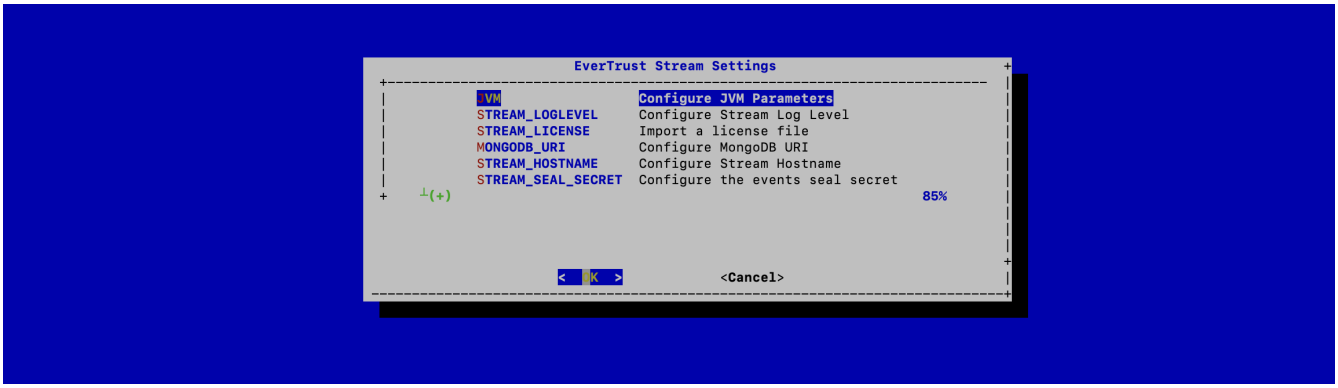
Start the Stream configuration utility by running:

```
$ /opt/stream/sbin/stream-config
```

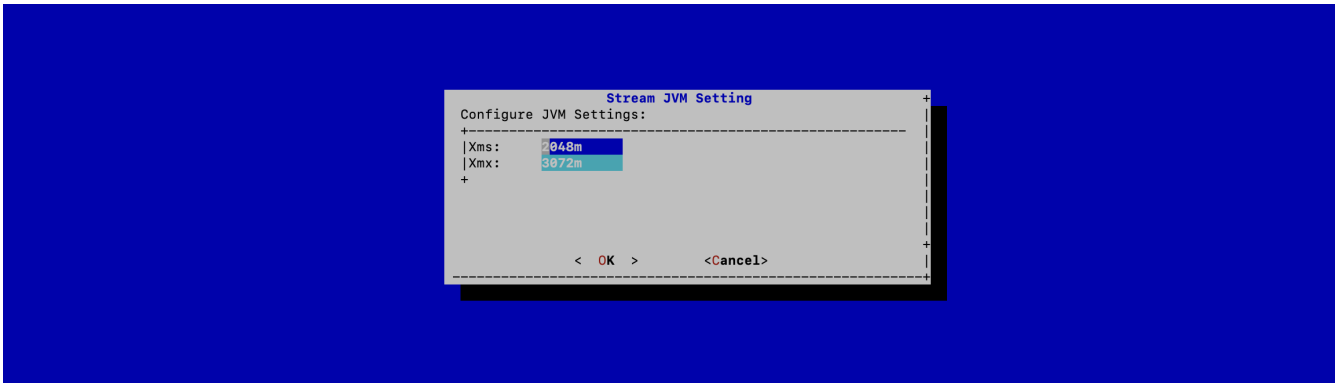
In the configuration menu, select **Stream**:



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
```

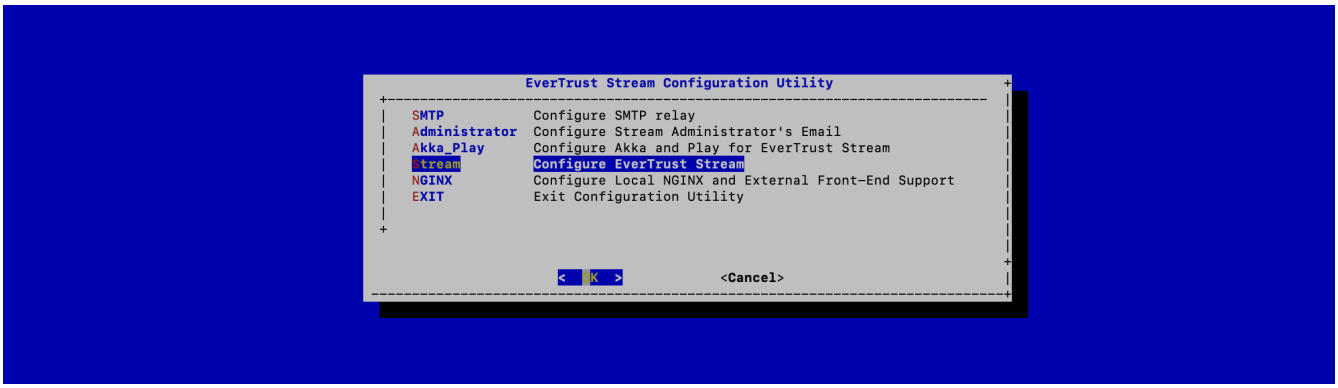
MongoDB URI Configuration

Access the server through SSH with an account with administrative privileges;

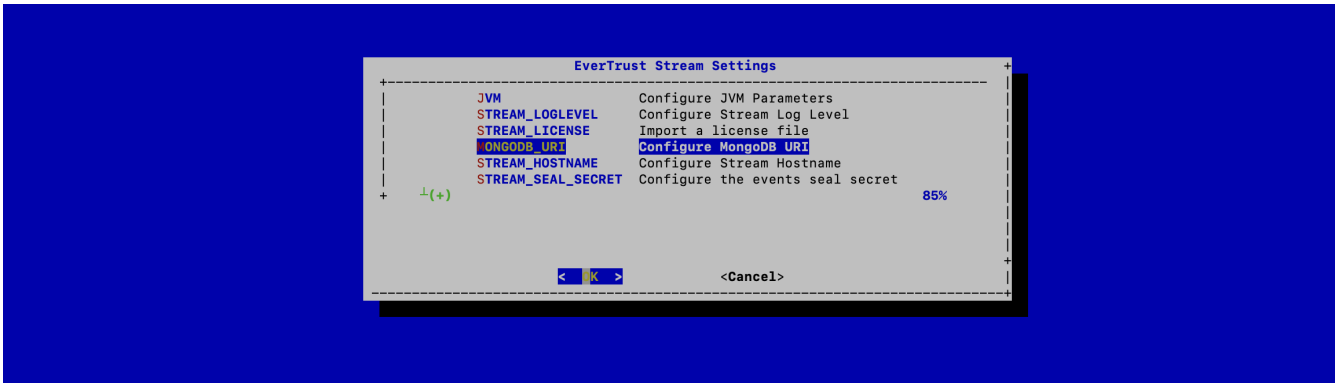
Start the Stream configuration utility by running:

```
$ /opt/stream/sbin/stream-config
```

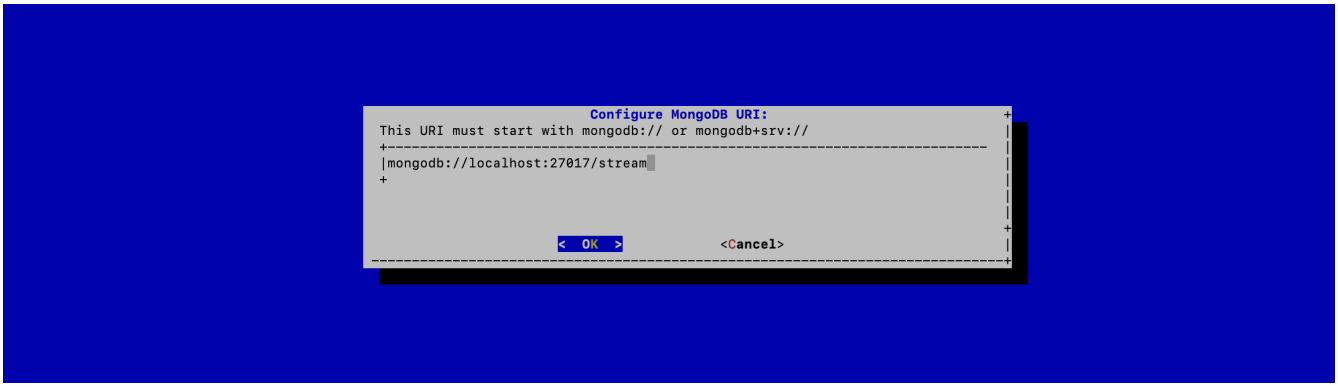
In the main menu, select **Stream**:



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



Specify the MongoDB URI to target your MongoDB instance:



NOTE	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+srv://<user>:<password>@<hostname>:<port>/stream
```

External MongoDB cluster of databases URI syntax

```
mongodb+srv://<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
```

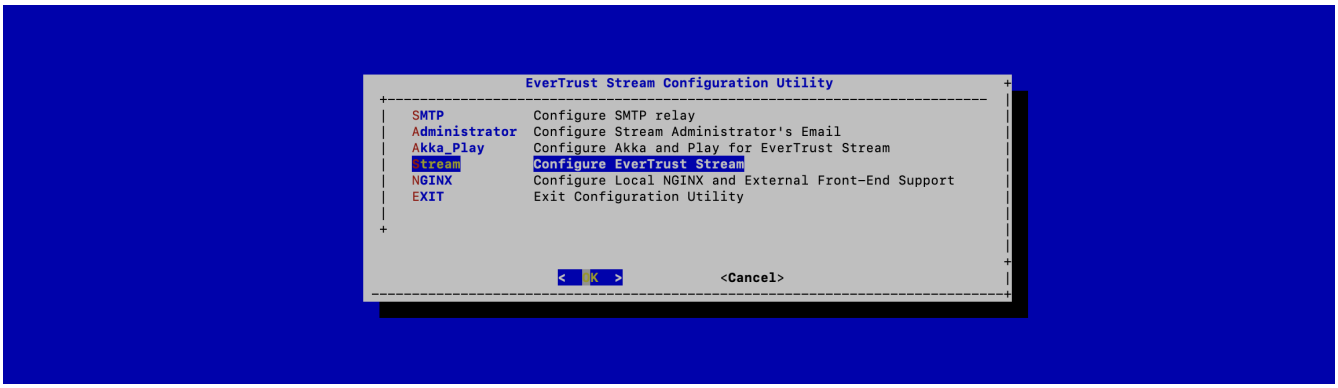
Stream Hostname Configuration

Access the server through SSH with an account with administrative privileges;

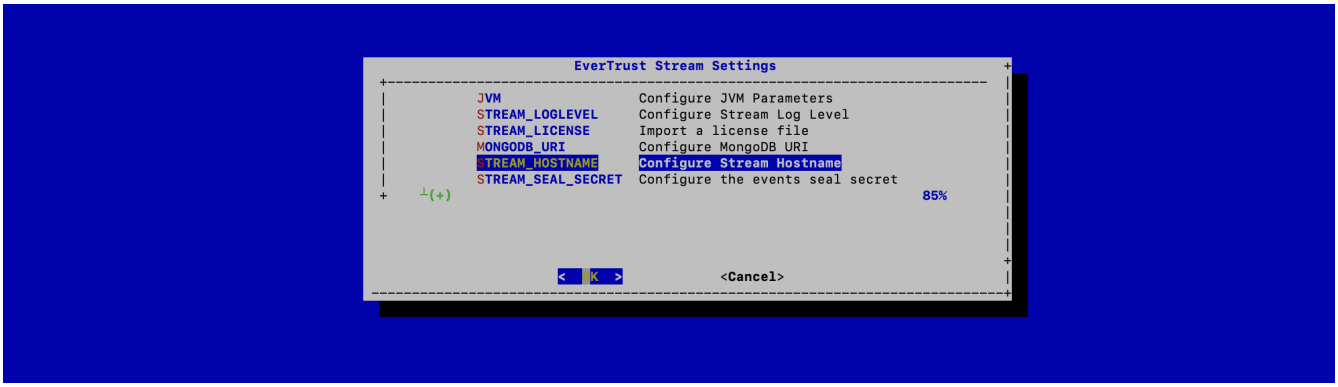
Start the Stream configuration utility by running:

```
$ /opt/stream/sbin/stream-config
```

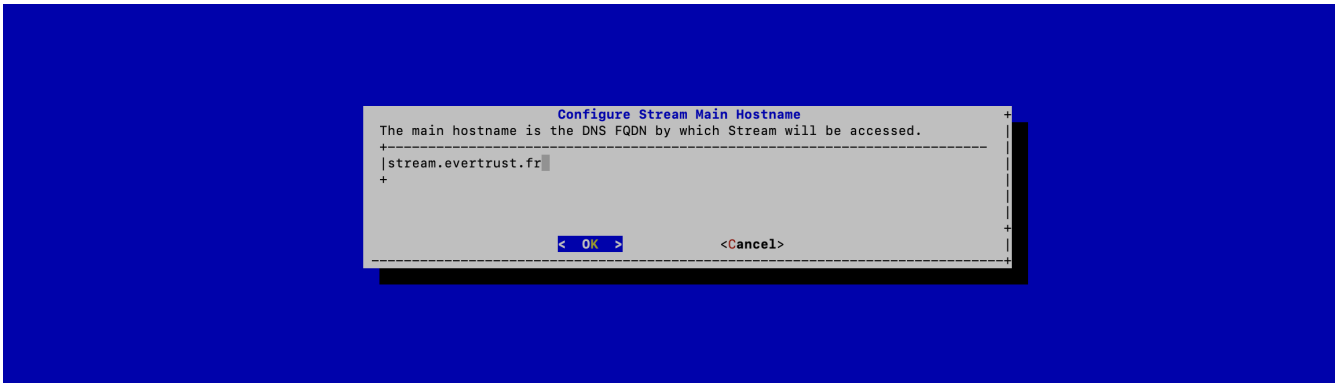
In the main menu, select **Stream**:



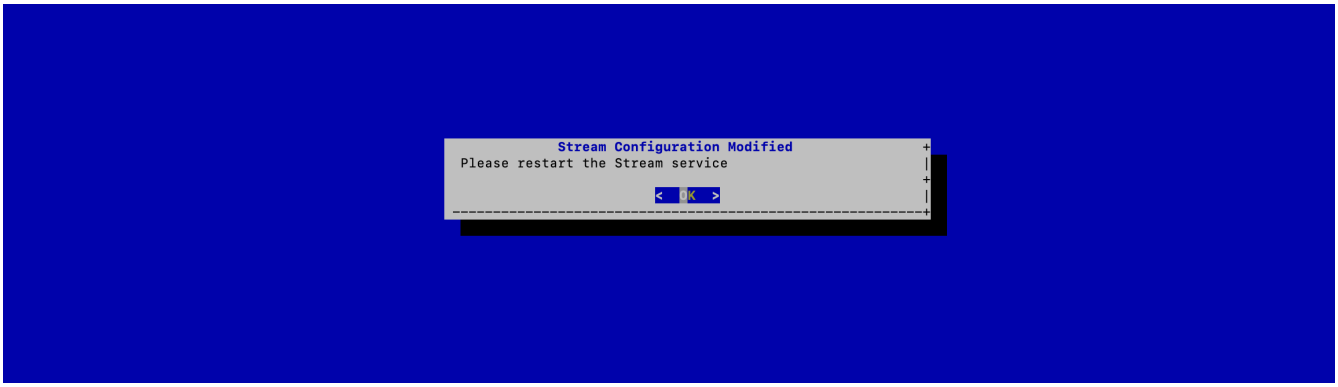
In the Stream configuration menu, Select **STREAM_HOSTNAME**:



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.

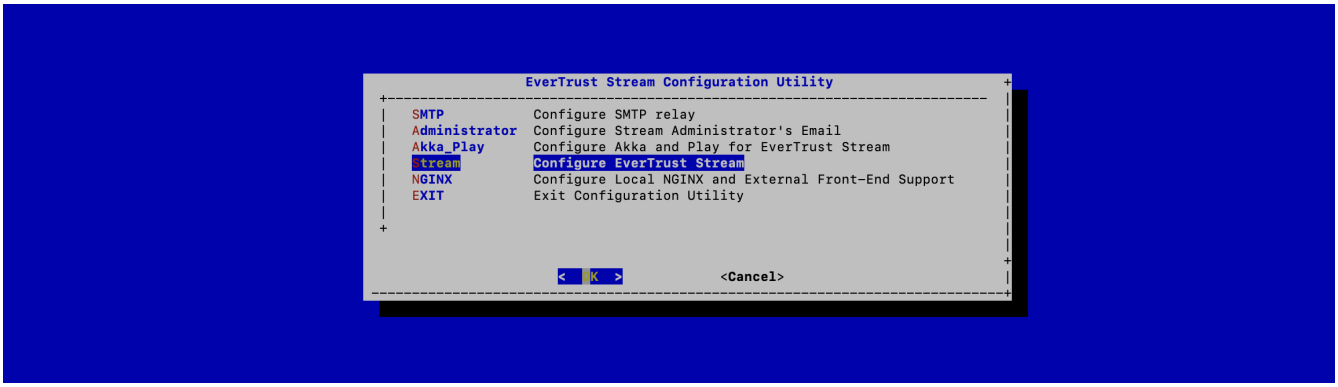
These events are typically signed and chained to ensure their integrity. Therefore, you must specify a sealing secret for this feature to work properly.

Access the server through SSH with an account with administrative privileges;

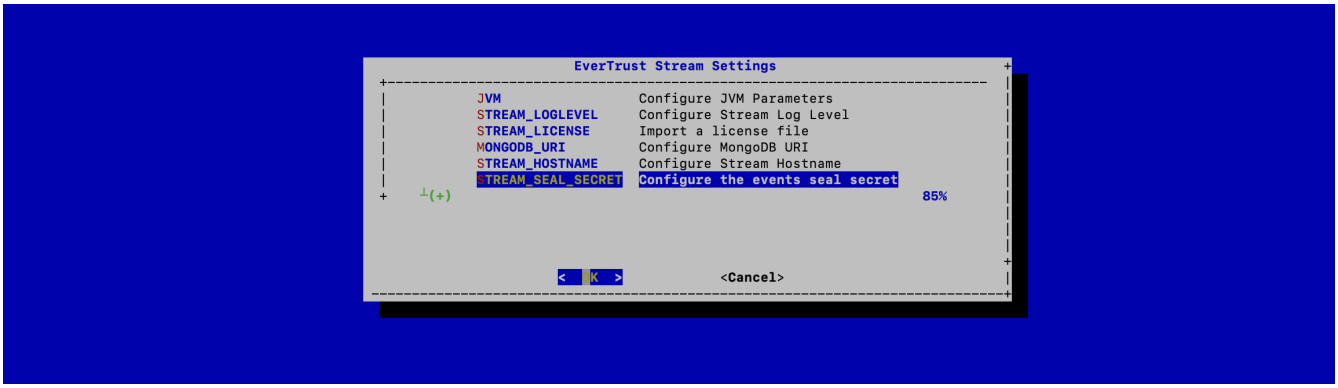
Start the Stream configuration utility by running:

```
$ /opt/stream/sbin/stream-config
```

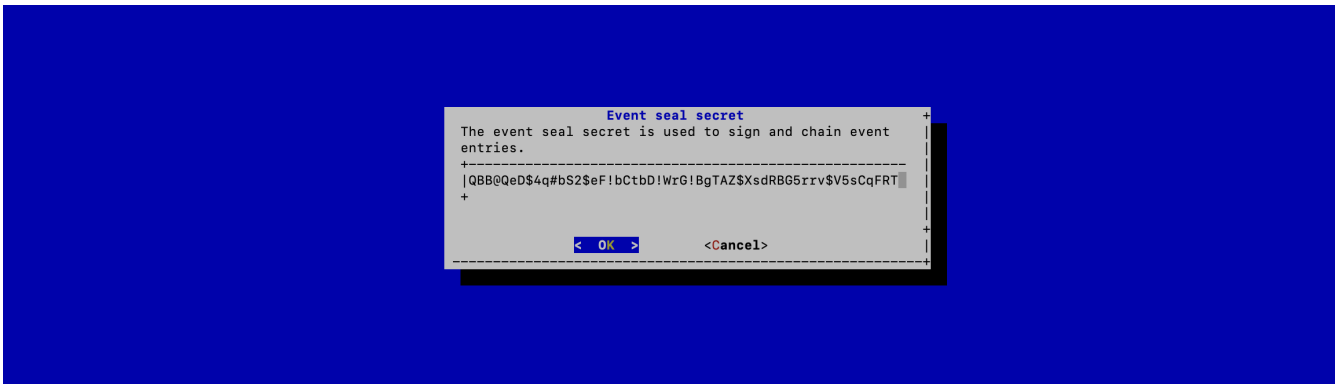
In the main menu, select '**Stream**':



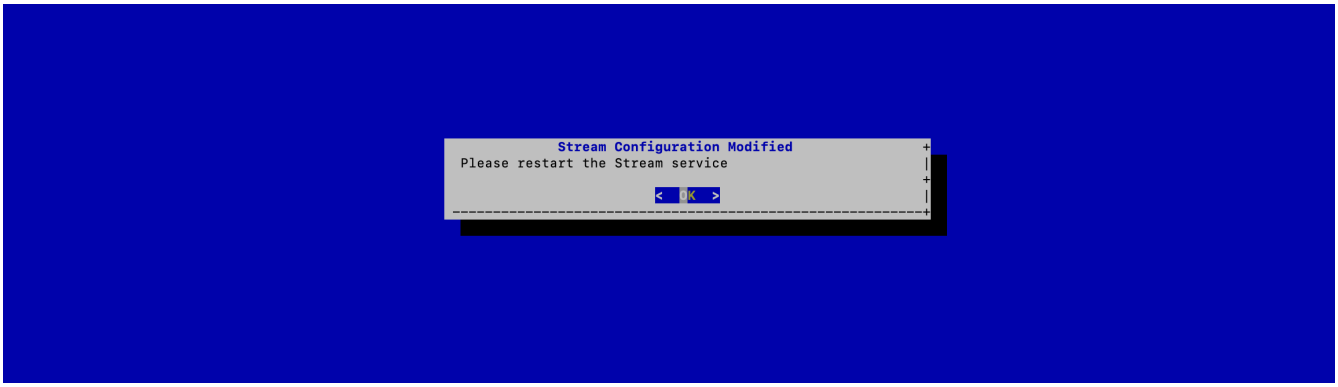
In the Stream menu, select '**STREAM_SEAL_SECRET**':



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

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

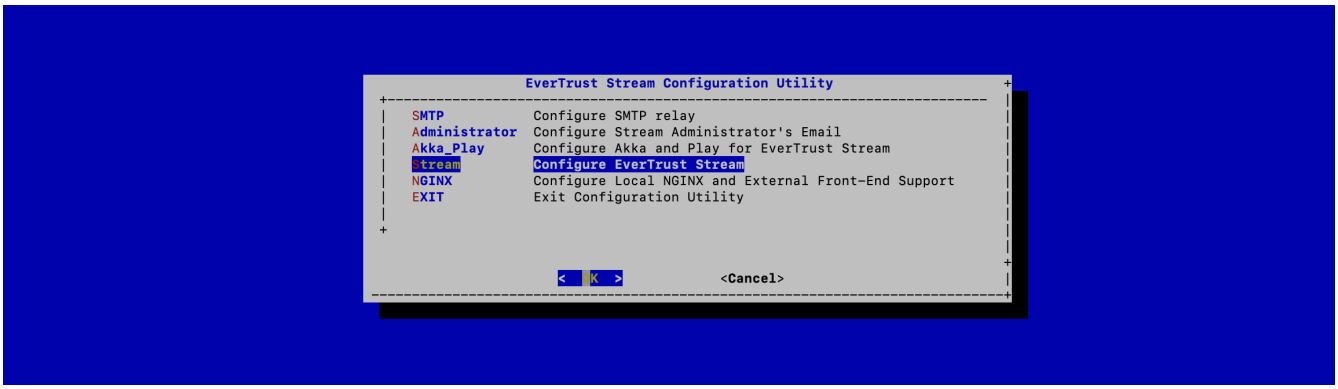
Upload the `stream.lic` file through SCP under `/tmp/stream.lic`;

Access the server through SSH with an account with administrative privileges;

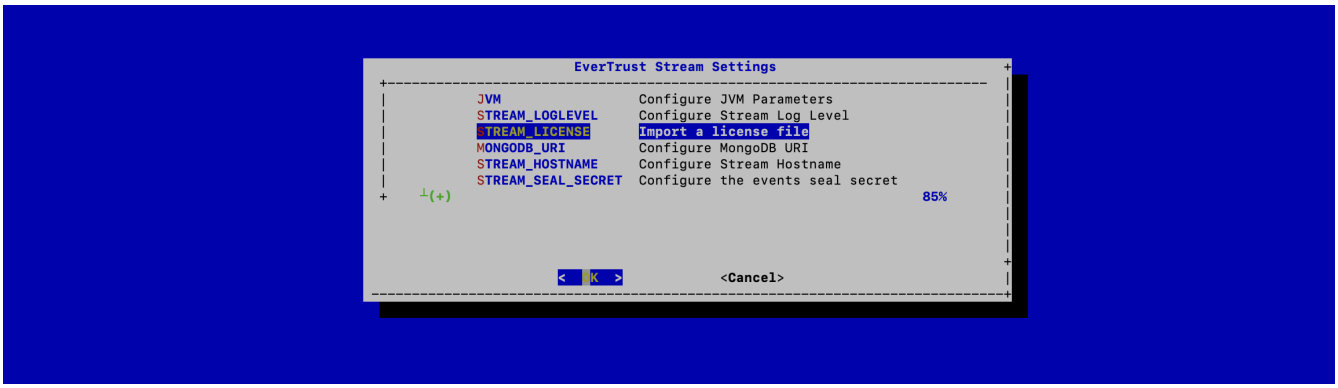
Start the Stream configuration utility by running:

```
$ /opt/stream/sbin/stream-config
```

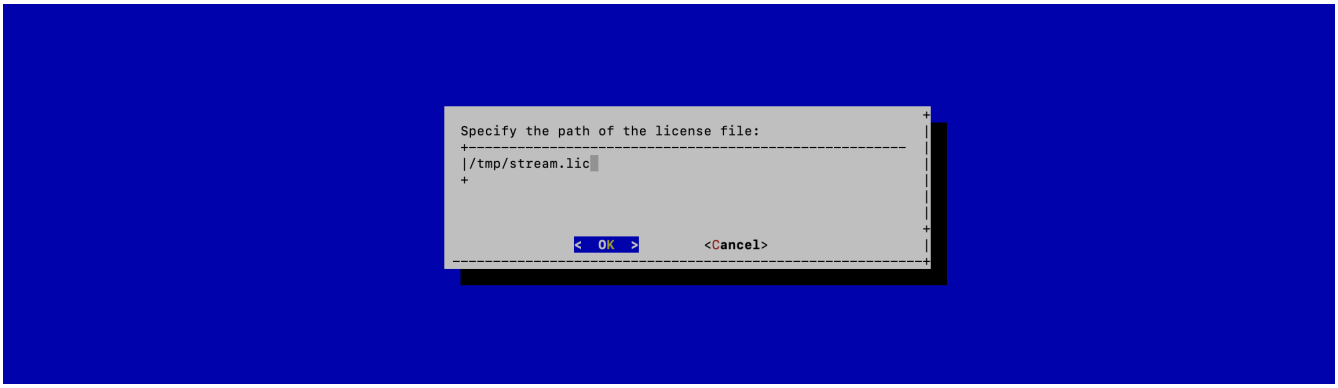
In the main menu, select **Stream**:



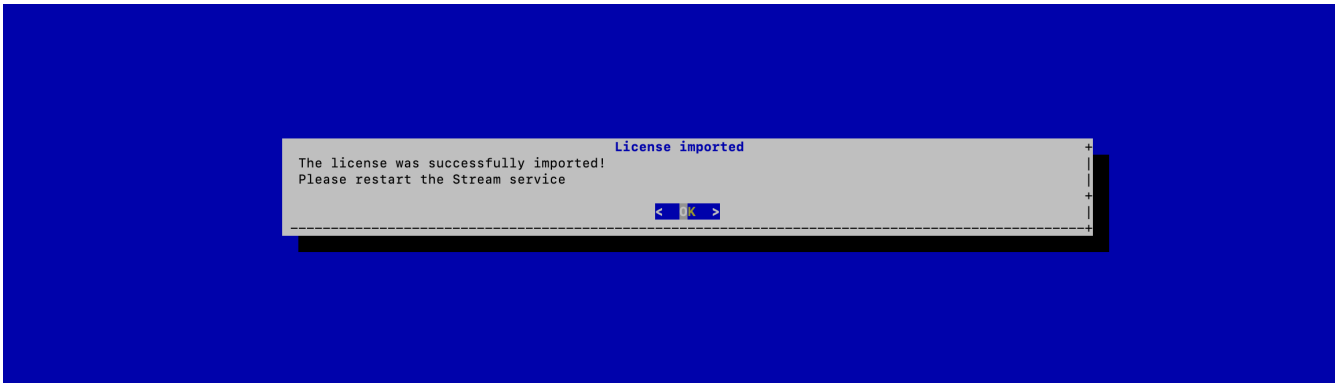
In the Stream configuration menu, Select **STREAM_LICENSE**:



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
```

2.3.2. Installing a Server Authentication Certificate

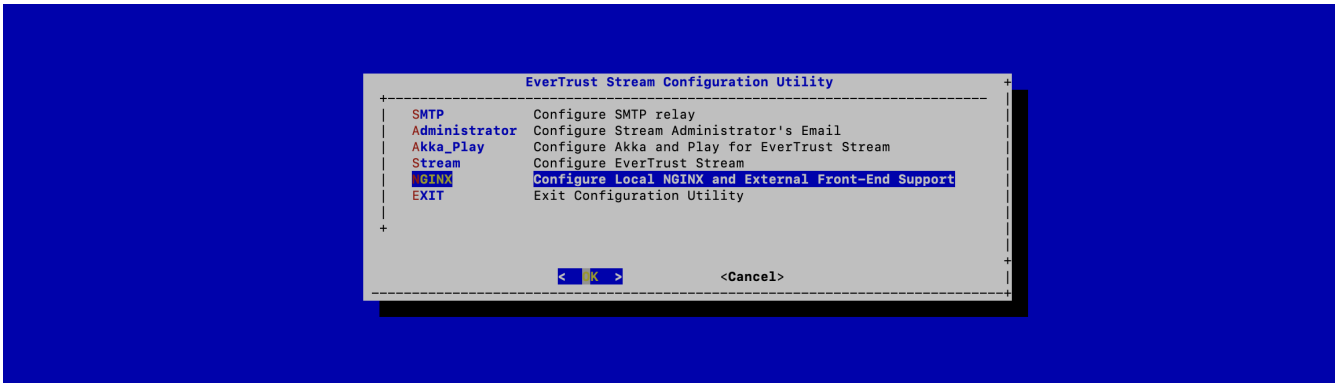
Issuing a Certificate Request (PKCS#10)

Access the server through SSH with an account with administrative privileges;

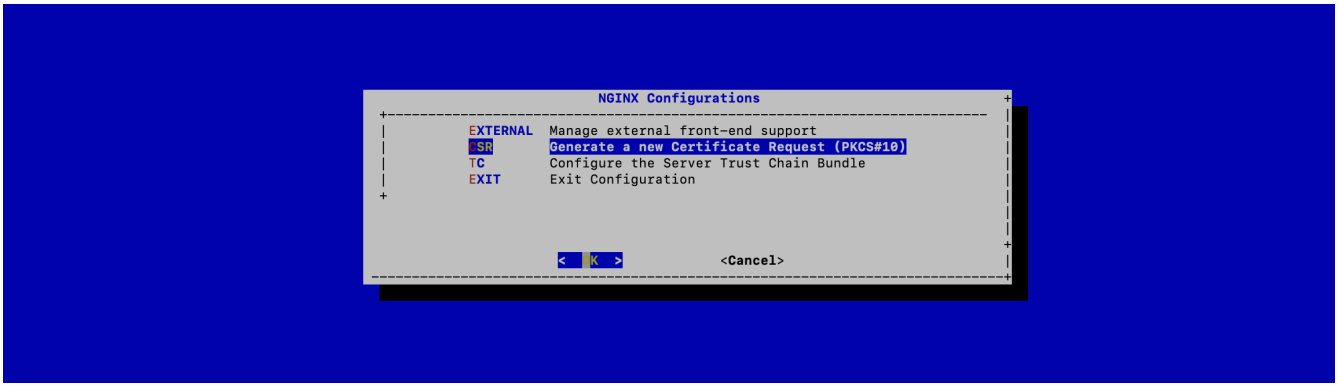
Start the Stream configuration utility by running:

```
$ /opt/stream/sbin/stream-config
```

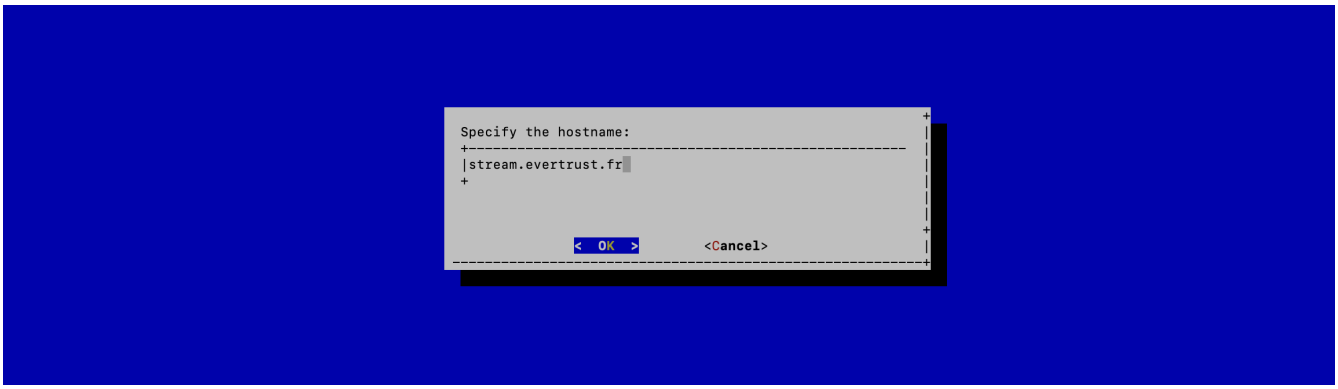
In the main menu, select '**NGINX**':



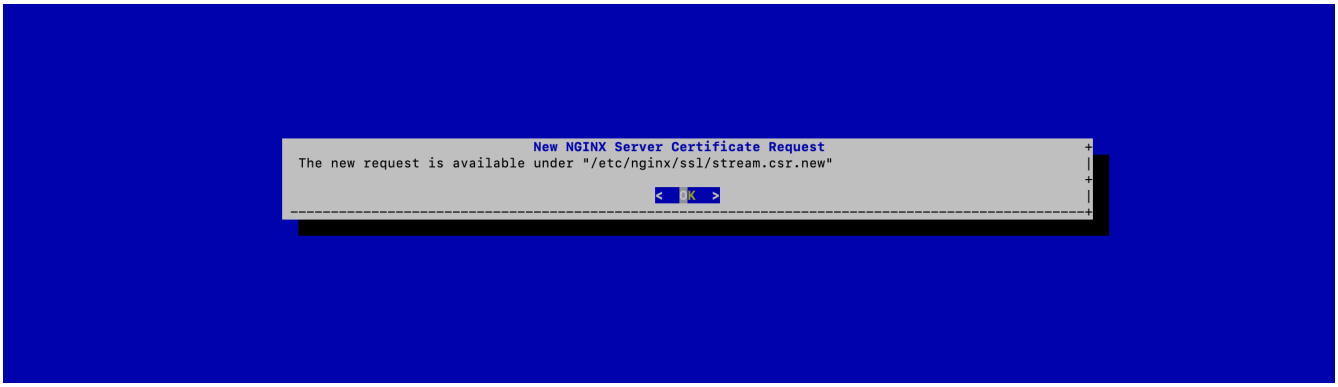
In the NGINX menu, select '**CSR**':



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 server certificate

Signing using an existing PKI

If you desire to sign your Stream web server certificate using an existing PKI, you need to provide your certificate authority with the `/etc/nginx/ssl/stream.csr.new` file that was generated at the previous step. You will then need to upload the signed certificate via SCP under `/tmp/stream.crt` (PEM and DER formats are supported).

Self-signing the certificate

If you plan on using the Stream PKI to manage the Stream web server certificate, you must self-sign it for configuration purposes, then refer to the administration guide to replace it later on.

To self-sign it using openssl, run the following commands:

```
# cd /etc/nginx/ssl
# openssl x509 -req -days 365 -in stream.csr.new -signkey stream.key.new -sha256 -out
/tmp/stream.crt
```

Installing the Server Certificate

Upload the signed server certificate (in PEM format) on the Stream server under `/tmp/server.crt` through SCP;

Access the server through SSH with an account with administrative privileges;

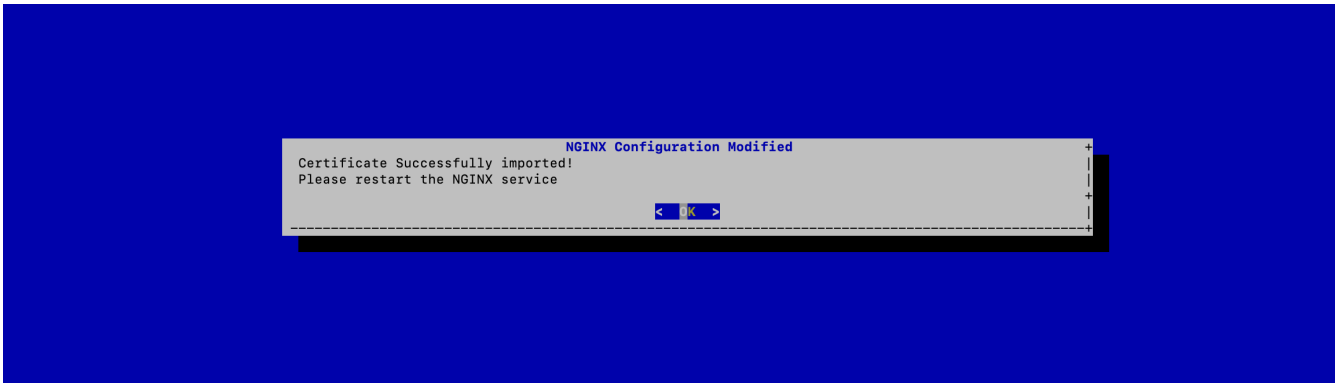
Start the Stream configuration utility by running:

```
$ /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

NOTE	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.

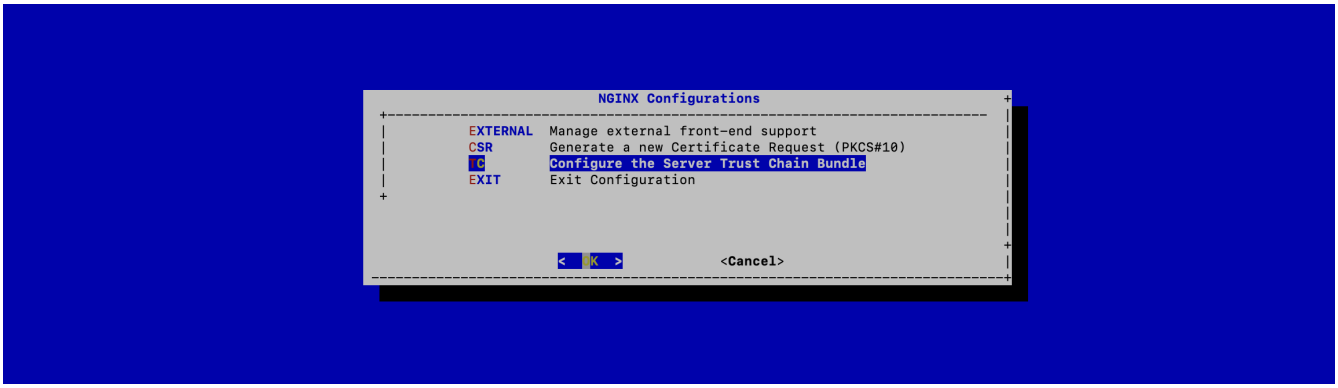
Upload the server certificate trust chain (the concatenation of the Certificate Authority certificates in PEM format) on the Stream server under `/tmp/server.bundle` through SCP;

Access the server through SSH with an account with administrative privileges;

Start the Stream configuration utility by running:

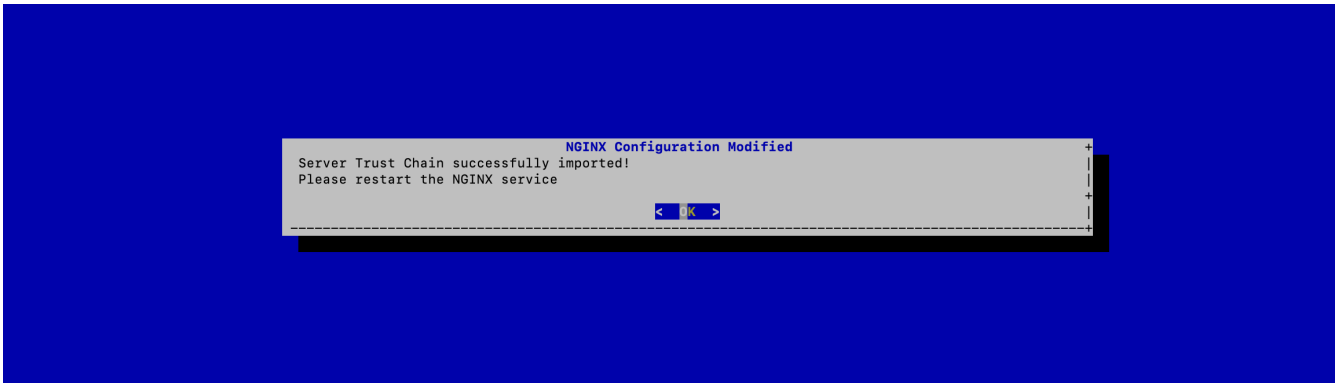
```
$ /opt/stream/sbin/stream-config
```

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



Specify the path `/tmp/server.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
```

Unresolved directive in index.adoc - include::pages/:iaas/access.adoc[leveloffset=+3] :leveloffset: +2

Upgrading

1. Upgrade the Stream installation

The first step in the upgrade procedure is to upgrade Horizon component itself.

1.1. If Stream was installed using a 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.

1.2. If Stream was installed manually

You must retrieve the latest Stream RPM from the EverTrust repository manually using the user credentials you were provided.

Access the server through SSH with an account with administrative privileges;

Install the Stream package with the following command:

```
# yum localinstall stream-1.1-1.noarch.rpm
```

2. 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.

Access the server through SSH 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 "<mongo uri>"
```

NOTE	The upgrade script requires a MongoDB client to connect to your database (either `mongo` or `mongosh`). If no client is 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.

2.1. Uninstalling

WARNING	Before uninstalling, please ensure that you have a **proper backup of the Stream component**. Once uninstalled, all Stream data will be **irremediably lost**!

NOTE

Uninstalling Stream consists in uninstalling:

- The Stream service;
- The MongoDB service;
- The NGINX service.

2.1.1. Uninstalling Stream

Access the server through SSH with an account with administrative privileges;

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
```

2.1.2. Uninstalling NGINX

Access the server through SSH with an account with administrative privileges;

Uninstall NGINX with the following commands:

```
# systemctl stop nginx
# yum remove nginx
# rm -rf /etc/nginx
# rm -rf /var/log/nginx
```

2.1.3. Uninstalling MongoDB

Access the server through SSH with an account with administrative privileges;

Uninstall MongoDB 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
```

3. Installing on Kubernetes

3.1. Installation

3.1.1. 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.

3.1.2. 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        0.1.3           1.1.1       EverTrust Stream Helm chart
```

3.1.3. 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

You should have both a license file (most probably named `stream.lic`) and a keyset for your Stream installation.

To generate a keyset, download our keyset utility onto a secure environment that has access to your cluster. Extract the archive and run the binary that matches your architecture. For instance :

```
$ ./tinkey-darwin-arm64 generate-keyset --out=keyset.json
```

Then, create a Kubernetes secret containing both files into the Stream namespace :

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

3.1.4. 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 :k8s/access.pdf. 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 k8s/production.pdf to make sure your instance is fit to run in your production environemnt.

Unresolved directive in index.adoc - include::pages/:k8s/access.adoc[leveloffset=+2] :leveloffset: +2

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.

1. Operating the database

All persistant 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 local MongoDB standalone instance will be spawned in your cluster, using the bitnami/mongodb chart. No additional configuration is required but it is not production ready out of the box. You can configure the chart as you would normally below the `mongodb` key :

  ```
  mongodb:
    architecture: replicaset
    # Any other YAML value from the chart docs
  ```

- 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.

2. 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
`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>
```

WARNING 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.

3. 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
```

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

4. 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
```

4.1. Upgrade

We recommended that you only change values you need to customize in your `values.yml` file to ensure smooth upgrading. Always check the upgrading instructions between chart versions.

4.1.1. Upgrading the chart

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        0.1.3           1.1.1       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 0.1.3
```

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.

WARNING	Before upgrading to specific chart version, thoroughly read any Specific chart upgrade instructions for your version.

4.1.2. Specific chart upgrade instructions

Empty section.

4.2. Uninstallation

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.

WARNING	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.

Unresolved directive in index.adoc - include::pages/:k8s/advanced.adoc[leveloffset=+2] <<< :leveloffset: +1

Troubleshooting

1. 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.

1.1. Checks performed

At the moment, Stream doctor checks for :

1.1.1. 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 (throws a warning if SELinux is enabled)
- 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.

1.1.2. 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)

1.1.3. 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 **AKKA 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 8558), 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 8558) 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).

1.1.4. 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.

1.2. 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 :

```
# stream-doctor -l
```

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

```
# stream-doctor -l 30
```

1.3. 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 :

```
# stream-doctor -o stream-doctor.out
```

1.4. 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.

1.5. Help menu

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