Flussonic Manual

Flussonic Media Server

Maksim Lapshin

Table of contents

1. Products

1.1 Flussonic Media Server

Flussonic Media Server is a server-side software for launching a high-load video streaming service of any scale. Flussonic Media Server can receive, store, transcode, and deliver video to any number of viewers on any device.

Flussonic is written in a programming language called Erlang, which achieves:

- parallel processing efficiency
- high fault tolerance of the server
- scalability of the service ranging from a single server to a complex distributed network

1.1.1 Fields of application

- IPTV and OTT services
- VSaaS and CCTV
- video streaming

1.1.2 Flussonic Media Server features

- **Live streaming**. Captures live from satellite, IP camera, capture card or video encoding program. Supports VMix, OBS, Atemi, HDMI-to-RTMP converters, video editing consoles and browsers. Supports SDI, ASI, SRT, RTMP, WebRTC, WebRTC with ABR and WebRTC playback (WHEP) with retransmitting settings. Plays multibitrate live and streams from the archive via HLS, DASH, or MSS over the Internet. Plays multibitrate live streams via WebRTC with low latency.
- **Captures from DVB capture cards**. Captures channels from DVB-T/C/S cards. Captures WebRTC (WHIP), H.323, UDP MPEG-TS, TCP MPEG-TS, and HTTP MPEG-TS, M4S/M4F.
- **Transmission to DVB networks**. Prepares MPTS for transmission to satellite according to the TR-101290 standards. Receives publishing MPTS via DVB-C card for transmission to DVB-C networks.
- **Reception of DVB subtitles**. Conversion of DVB subtitles from pictures to text.
- **Video on Demand (VOD)**. File distribution with adaptive bitrate and language selection.
- **Own channel from VOD files**
- **DVR archive**. Records video streams to disk and manages the archive, supports the required archive depth and hard disk capacity. Plays multibitrate DVR via HLS, DASH, and MSS. DVR API.
- **Delayed playback**. Plays multibitrate live streams with delay via HLS, DASH or MSS.
- **DRM**. Supports Widevine, PlayReady, Solocoo, Conax, EzDRM, BuyDRM KeyOS, and other DRM systems. See Flussonic API Reference for a complete list.
- **Cloud storage**. Fetches files from HTTP servers and cloud storages like Amazon S3 and OpenStack Storage (Swift), and stores the archive in cloud storages.
- **Transcoder**. Supports H.264, H.265, AV1, MPEG-2 video, deinterlacing, multibitrate, AAC, MP3, Opus, MPEG-2 audio, AC3, PCMA, PCMU audio codecs.
- **CDN integration**
- **Plays MPTS multicast and unicast**
- **Delivers SPTS to multicast group for local networks**
- **Web player for HLS, DASH, MSS and WebRTC**
- **Ad Insertion**. Server-side ad insertion.
- **Source capture failover using multiple servers**.

- **API for integration**. Integration with Middleware and service website.
- **Channel viewing statistics**
- **Guides for solutions to typical problems**. Capture, transcoding, archiving with access, packager, player, monitoring.
- **Project Support**

1.1.3 Start working with Flussonic Media Server

To start working with Flussonic Media Server, do the following:

1. Request a trial by filling out the trial form on our website
2. Follow the steps in the Quick start with Media Server.

What's new

Find the changes added in the latest Flussonic versions in our blog.

2. Manuals & Guides

2.1 Media server quick start

The primary goal of this quick start tutorial is to introduce you to Flussonic Media Server. By the end of this tutorial you will learn how to:

- Install Flussonic Media Server
- Configure and view a video stream
- Publish video to Flussonic
- Upload and view a video file
- Install Flussonic on a Digital Ocean Server

Note

In this documentation, we will use placeholder IP addresses (or URLs) of the Flussonic server (such as FLUSSONIC-IP). Please replace placeholder IP addresses with the actual IP addresses used on your server.

2.1.1 Flussonic Media Server

Flussonic Media Server is software for video streaming server capable of a wide variety of tasks including mass storage, transcoding, live and on-demand video delivery and control over video consuming and video streams.

We will demonstrate all main scenarios using Flussonic web interface. However, if you prefer to use API, please refer to Flussonic API reference.

2.1.2 Installing Flussonic Media Server

Installation

This section briefly describes how to install Flussonic Media Server so that you can install it quickly.

To install and configure Flussonic Media Server, you will need a computer with Linux connected to the Internet, and a license key. Contact our manager to purchase a license.

The main requirement is that the system must be 64-bit. We strongly recommend using Ubuntu Server. You can find the whole list of system requirements here.

Note

Despite the fact that Flussonic Media Server will work on Ubuntu Desktop, we do not recommend using it, because Ubuntu Desktop has its own features with power management, energy saving, Network-manager and background updates, and other differences that may affect on performance. It is also possible that some third-party software and drivers may not work on it.

If you don't have an available suitable system at hand, you could rent a small cloud instance at Digital Ocean for the time needed to try out our software. If you're not sure how to do this, we have detailed instructions to help you.

To install Flussonic Media Server you will need access to a Linux console as the 'root' user on your server.

Run the following command in the Linux console (command line):

```
curl -sSf https://flussonic.com/public/install.sh | sh
```

Then start Flussonic Media Server:

```
service flussonic start
```

Now you can open the Flussonic administrator's web interface in a web browser.

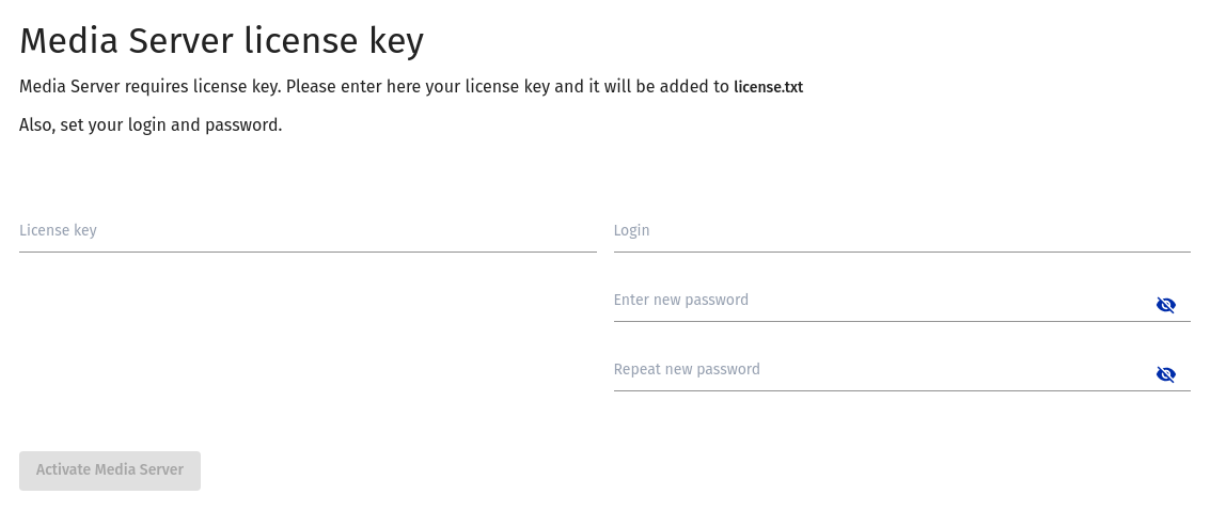
The first run of the Flussonic user interface (UI)

The Flussonic user interface is available at `http://FLUSSONIC-IP:80/` (replace `FLUSSONIC-IP` with the real IP address of your server).

On the start page Flussonic asks you to enter the administrator's username and password and the license key that you have received.

> **Warning**
>
> Permanent Internet access is required for activation with the license key and continuous use of *Flussonic Media Server*. Learn more at Using the License Key.

> **Warning**
>
> spec_chars_note_en



Checking the installation

You can check whether your Flussonic installation is correct by visiting `http://FLUSSONIC-IP:80/`, where `FLUSSONIC-IP` is the address of the server on which you installed the software. The Flussonic administrator's web interface opens if the installation was correct.

If the web interface failed to open, please check the details in the Installation section or contact Flussonic technical support.

See also:

- The detailed instruction on how to install the software can be found in the Installation section.

2.1.3 Live streaming

Flussonic can receive streaming video in two main ways: acting as a client or a server.

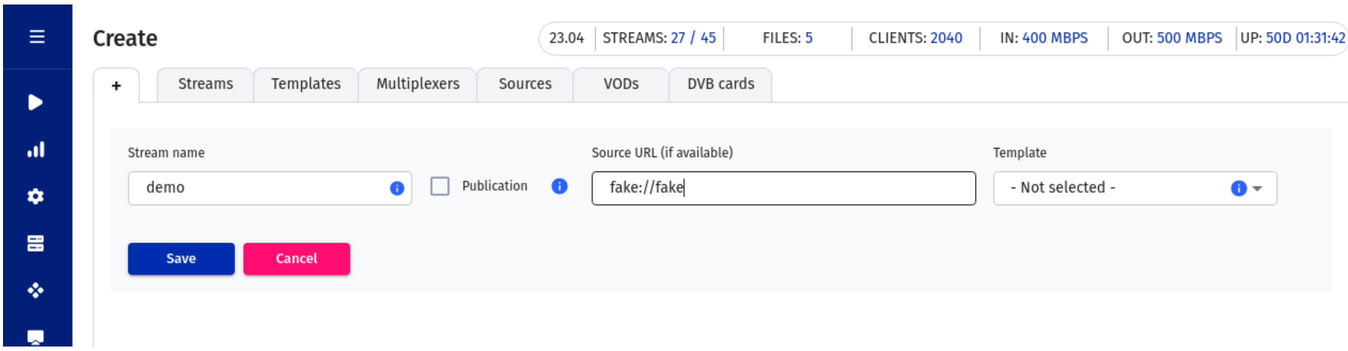
When acting as a client, Flussonic connects to a video source to retrieve (*ingest*) the data. When acting as a server, Flussonic waits for external systems to connect and then it receives video for *publication*.

Ingesting a stream

A video source can be an IP camera, other video streaming server, a specialized program working with a DVB card, and almost any system that can stream video over the network. Flussonic supports all major video transfer protocols.

In addition, Flussonic can generate a sample video stream `fake://fake`. This stream can be used, for example, to test the system health.

To add a live stream, go to **Media** > click **Add stream**. Specify a stream name (`demo`) and a source URL (`fake://fake`). Click **Create**.



Now open the address `http://FLUSSONIC-IP:80/demo/embed.html` in the browser and see the result.

See also:

- Learn more about playback URLs here.
- Learn more about live video in the Live streaming section.

Accepting a published video

Publication is a process where an external system connects to Flussonic Media Server and initiates the transmission of streaming video to Flussonic Media Server. To make this possible, you will need to configure a stream or a publishing location on the Flussonic server where you allow publication.

The publishing location can have *static or dynamic name*:

- **Static** names are enough when you have one stream from one source that is published fairly permanently. With static name, it is enough to specify a special `publish://` option as a source URL when creating a stream in Flussonic. In the publication source, specify one of the links from the **Publish links** section on the **Overview** tab in the profile of the created stream.
- **Dynamic** names are useful when you have many changing publication sources and you don't know for sure how many and which streams you need to accept. You will need to configure a *template* with a publication *prefix* where you allow publication. A single publishing location will be used to publish one or more streams. The prefix is used to form a stream name. The general structure for a stream name is as follows: `http://FLUSSONIC-IP:80/PREFIX/STREAM_NAME`.

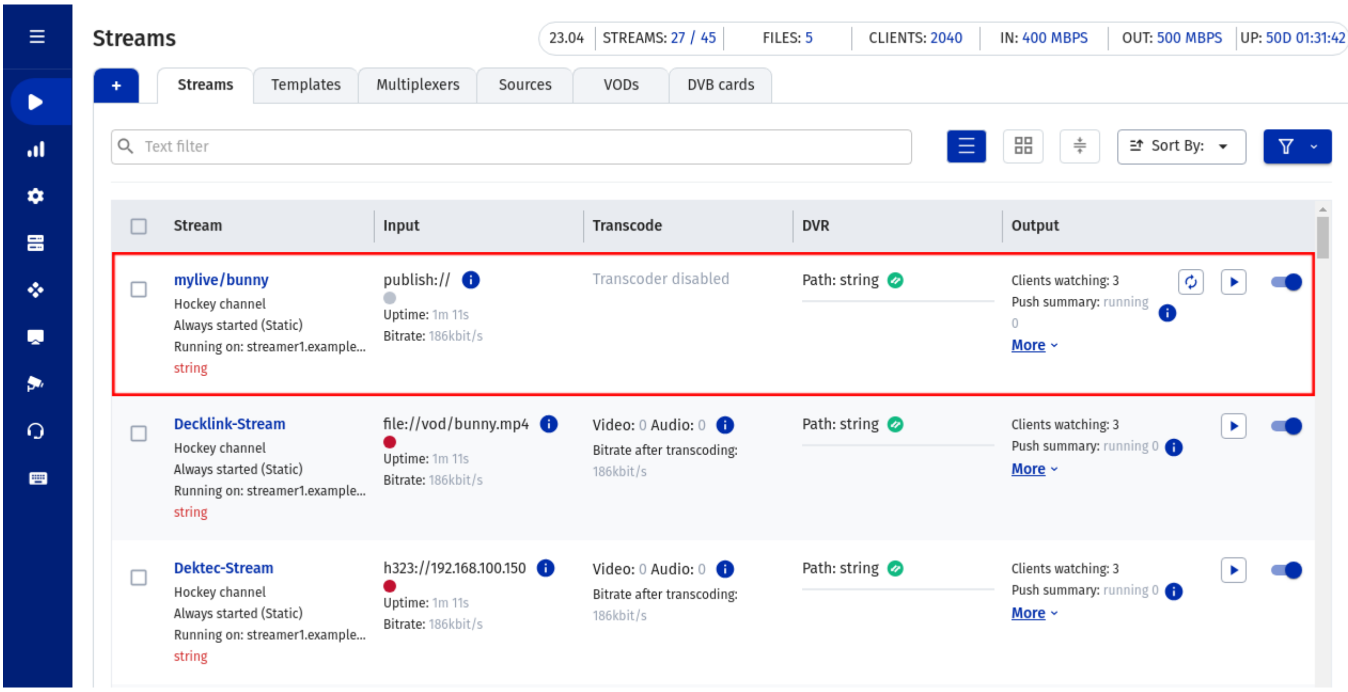
Let's configure a publication with dynamic name:

1. To create a template go to **Media** > **Templates** > click **Add template**. Specify a template name (for example, `live-mylive`) and a special `publish://` option as a source URL. Click **Create**.
2. Then click the name of the created template, and in **Template settings** specify a prefix (`mylive`). Then click **Save and apply to streams**.
3. Go to publication source (external app) to set the stream URL. If you configured the prefix `mylive`, then you must specify the stream name that starts with `mylive/` in the URL, for example, `mylive/bunny`.

   Let's transmit video by using the RTMP protocol. We will use the file `/opt/flussonic/priv/bunny.mp4` as a source (this file is already included into the distribution package). Run the following command:

```
/opt/flussonic/bin/ffmpeg -re -i /opt/flussonic/priv/bunny.mp4 -c copy -f flv rtmp://FLUSSONIC-IP:1935/mylive/bunny
```

Publishing will start. On the **Media** tab, you can see a stream for publishing that is automatically generated from the template:



To watch the stream, open this address in the browser:

```
http://FLUSSONIC-IP:80/mylive/bunny/embed.html
```

See also:

- Refer to the Publishing section to learn more about publishing video streams to Flussonic.

2.1.4 File playback

In this section you will learn how to play a video file using Flussonic. For playing files, Flussonic uses *VOD (Video On Demand)* service — an integral part of services based on video delivery. To play a file, you will need to:

1. Set up a *VOD location* to specify how the path in requests for the file playback should match the real file on the disk or in an HTTP repository. To add a VOD location, go to **Media** > **VODs** > click **Add VOD** > enter **VOD name** (for example, `Movies`) and **File directory path** (`/storage`) > click **Create**.

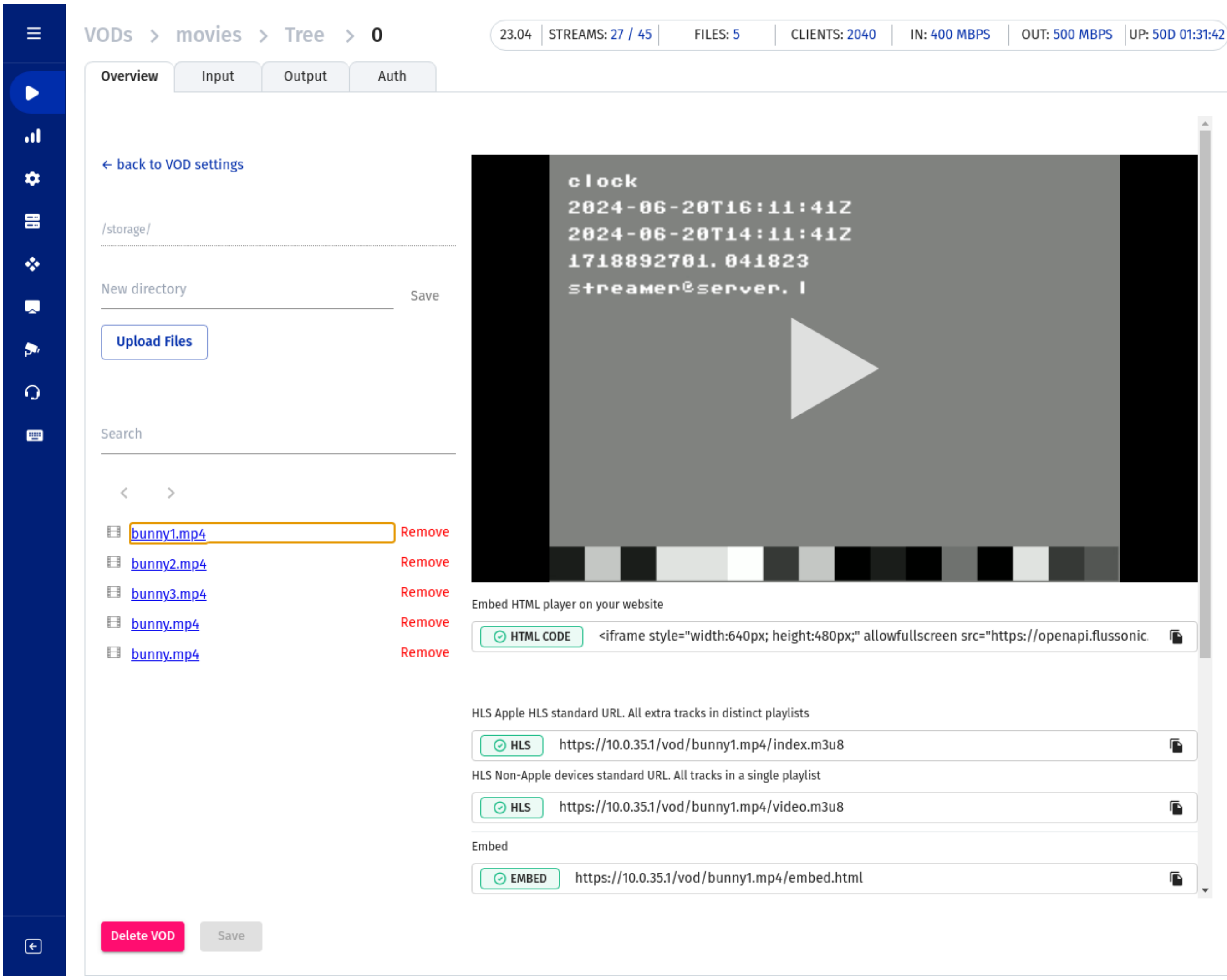
   Now Flussonic knows that when clients request `/movies/bunny.mp4`, it will need to access the file `/storage/bunny.mp4`. In other words, everything after the prefix `movies` will be cut and added to the specified path on the disk (that starts with `/storage` in our example).

2. Now you can upload the file to the `/storage` directory. Go to **Media** > **VODs** > click the name of the created VOD location (`Movies`) > click **browse** > click **Upload Files** > select the file to upload (`bunny.mp4`).

   > **Note**
   >
   > The Flussonic distribution package includes a test file `/opt/flussonic/priv/bunny.mp4`, so you may just copy it to the `/storage` directory or download the freely available Big Buck Bunny video clip.

3. Open this link: `http://FLUSSONIC-IP:80/movies/bunny.mp4/embed.html` to check how the file is played.

To view all other links for playing the file, go to **Media** > **VODs** > click the name of the created VOD location (`Movies`) > click **browse** > click the name of the file. You will see the embedded player for playing the file, the HTML code for using in a player on your website or in your application, and the list of the links for playing the file via various protocols.



See also:

- Learn more about video files in the VOD Files page.

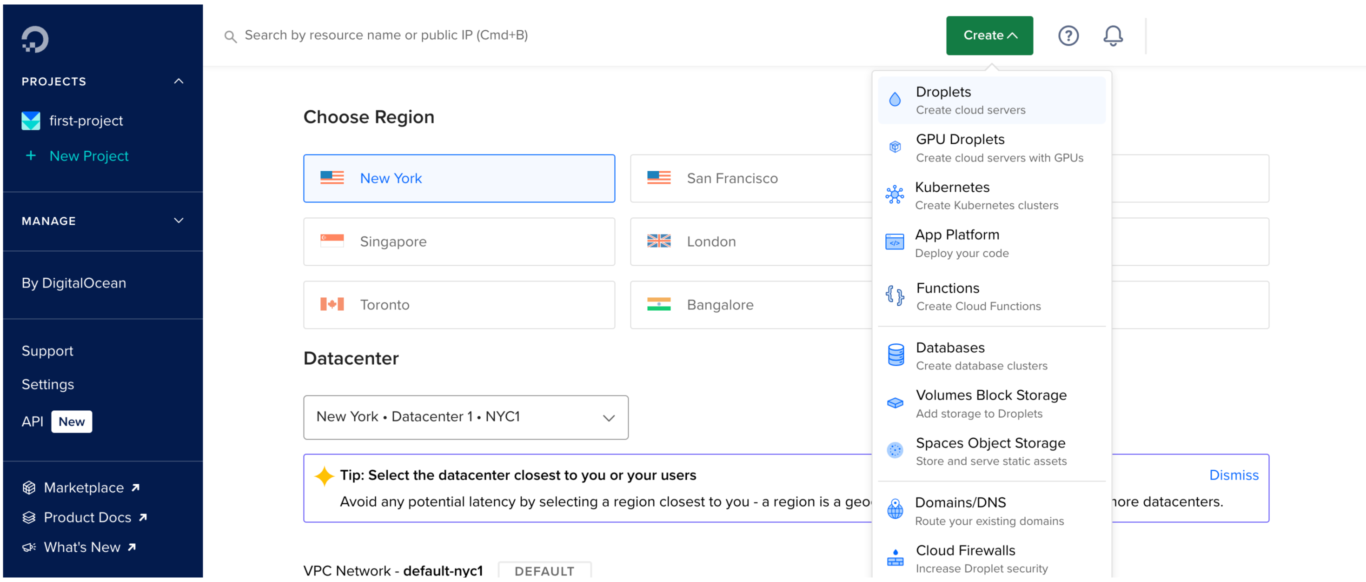
2.1.5 Install Flussonic Media Server on a Digital Ocean Server

In this guide, we will show you how to create an instance on the Digital Ocean platform and deploy Flussonic on it. This will take you about 7 minutes, even if you are using the computer console for the first time.

Creating a Cloud Server

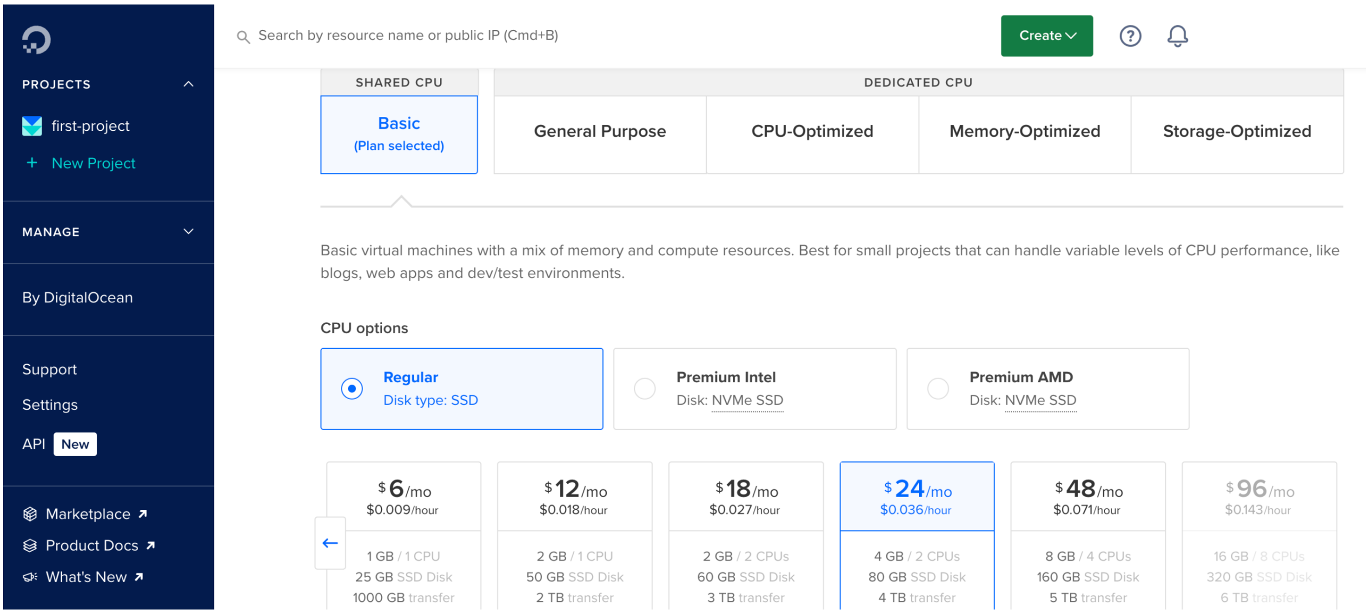
Create an account on Digital Cloud website.

In your personal account, select **Create** > **Droplets**.



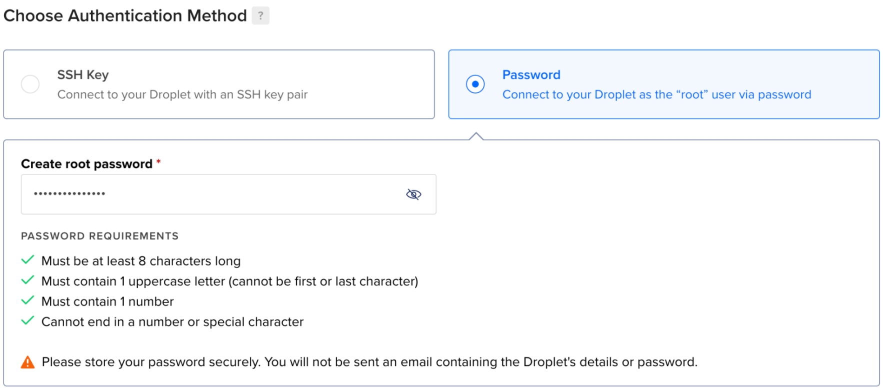
Configure the Server. For testing Flussonic’s capabilities, we recommend the following settings:

```
1. Operating System: Ubuntu 24.04 LTS 64-bit
2. CPU options: Regular, SSD disk, 2 CPU, 4 GB
3. Leave the other fields at their default values.
```



Choose Authentication Method

When creating the server, you also need to add an SSH key or a password. Use the password option.

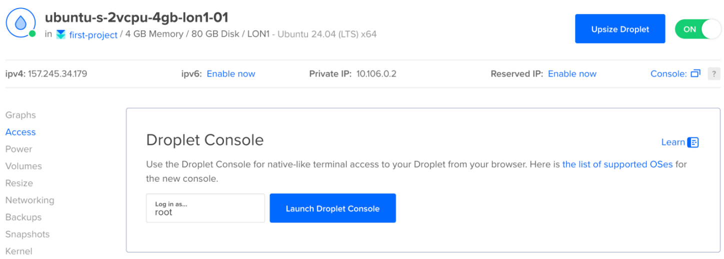


Next, you will see the cost of the tariff for the server with the selected configuration. You can view both the hourly and monthly prices. Digital Ocean charges only for actual usage: if you create a server, test Flussonic for two hours, and then delete the server, you will only be charged for two hours.

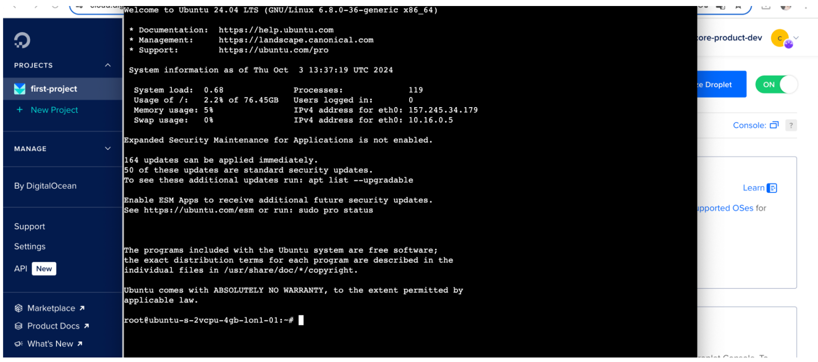
Create the server.

Downloading and Launching Flussonic

1. Click on the droplet you created. Open the **Access** option and click the **Launch Droplet Console** button.

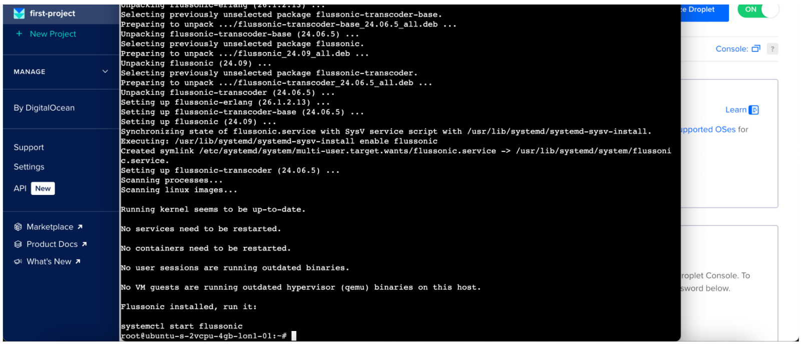


2. Open the Server Console:



3. Insert the following command to download Flussonic Media Server: `curl -sSf https://flussonic.com/public/install.sh | sh`

   Flussonic will begin downloading.



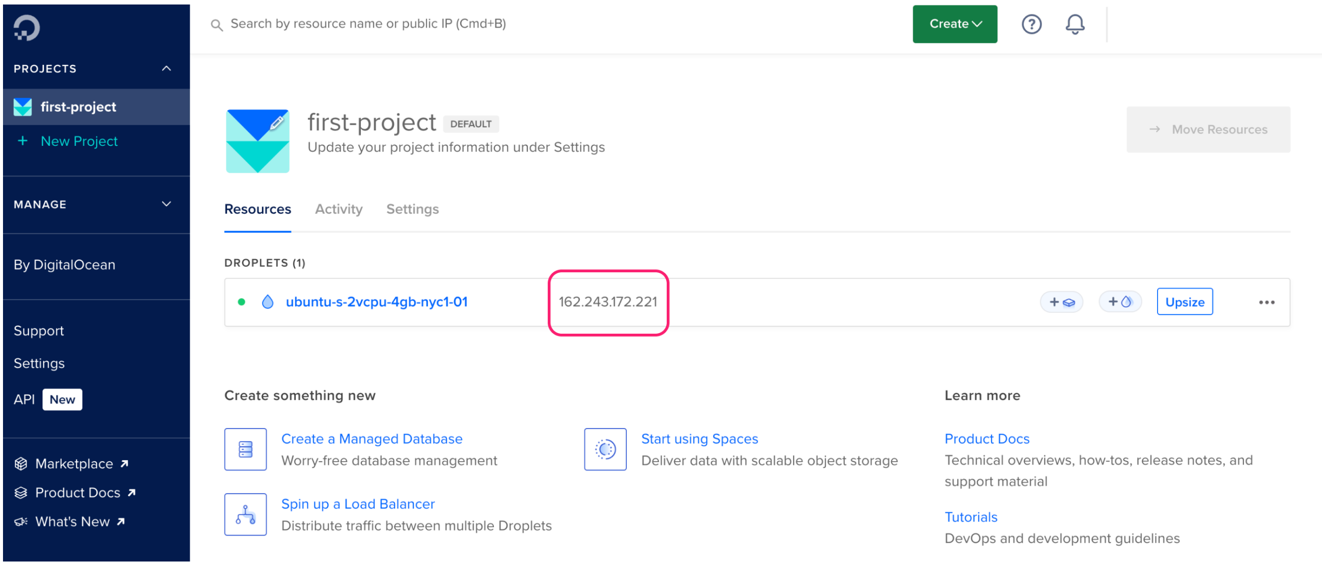
4. After the message 'Start Flussonic' appears, enter the command: `service flussonic start`

The server is now running.

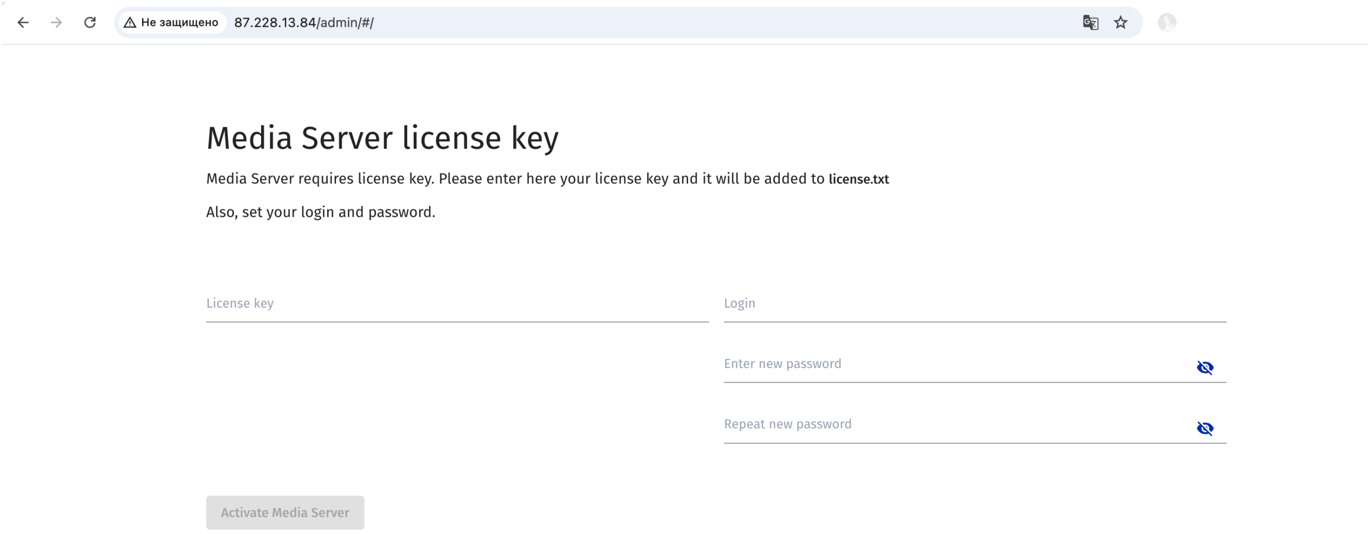
Activating the License Key

Open the Flussonic web interface. In your browser, go to 'http://FLUSSONIC-IP:80/', replacing 'FLUSSONIC-IP' with your server's IP address.

To find the IP address, open your Digital Ocean project with the droplet you created. The address is in the line.



On the Flussonic page: You will be prompted to enter a username, set an administrator password, and input the license key you received in your personal account.



To start your first video streams, proceed to the Quick Start Guide.

2.2 Administrator

2.2.1 Starting

System requirements

Please see the minimum system requirements to the host server for running *Flussonic Media Server* in the table below. In reality, the requirements may slightly vary depending on the number of concurrent connections to *Flussonic* server.

> **Warning**
>
> When calculating host server capabilities, all resources required for normal functioning of the operating system and other services running parallel to *Flussonic* must be taken into account.

MINIMUM SYSTEM REQUIREMENTS

Concurrent connections	10	100	1 000	5 000+
Processor	Any	Single core	Quad core (Xeon/Core i7)	Dual core Xeon E5
RAM	128 MB	256 MB	1024 MB	16 GB
Free disk space	40 MB	40 MB	40 MB	40 MB
Network adapter	100 Mbit/s	1 Gbit/s	1 Gbit/s Server NIC	10 Gbit/s Intel
Operating system	Ubuntu Linux			

For stable streaming video playback with a high volume of concurrent connections, we recommend distributing the traffic load among several real servers. For detailed information on clustering of *Flussonic* servers, please see the Clustering section.

Please note that when files on disk are used as the data source, the disk subsystem bears the main burden. Consequently, when planning the host server architecture for running *Flussonic Media Server*, special attention should be paid to the hard disk performance. For more detail on this subject, please see file streaming.

Firewall protection is not recommended, consider contacting support if you still need it.

BROWSER REQUIREMENTS

Recommended for using Fussonic Media Server:

- Mozilla Firefox 70 or higher
- Google Chrome 79 or higher

Not recommended (Flussonic Media Server can function with some restrictions):

- Microsoft Edge 80 or higher
- Safari 13 or higher

Also make sure that the browser is supported by its manufacturer (not EOLed).

Installing Flussonic

Learn how to install, activate, and start Flussonic Media Server.

On the page:

- Prerequisites
- Installing Flussonic Media Server
- Activating Flussonic Media Server
- How to change the administrator's password?
- Starting and stopping Flussonic Media Server
- Running Flussonic in a Docker container

PREREQUISITES

Before you install Flussonic Media Server, make sure the following conditions are met:

- Your system satisfies the system requirements.
- HTTP port 80 is open and no other applications in your OS listen to this port. By default, Flussonic Media Server uses HTTP port 80.

INSTALLING FLUSSONIC MEDIA SERVER

You can install Flussonic Media Server on Ubuntu, CentOS/RedHat and other RPM-based distributions.

On Ubuntu

Supported architectures: amd64, arm64.

Supported OS versions: Ubuntu LTS 24.04, 22.04.

Install Flussonic Media Server using the `apt` tool:

```
curl -sSf https://flussonic.com/public/install.sh | sh
```

After installation finishes, activate Flussonic Media Server

On RPM-based CentOS/RedHat and others

Danger

We strongly recommend that you avoid using RPM-based distributions like CentOS, RedHat, and Suse. We do not provide technical support on issues concerning RPM packages and distributions to users with less than 10 licenses.

Install Flussonic Media Server from Yum repository using the following command in the terminal:

```
cat > /etc/yum.repos.d/Flussonic.repo <<EOF
[flussonic]
name=Flussonic
baseurl=http://apt.flussonic.com/rpm
enabled=1
gpgcheck=0
EOF
yum -y install flussonic-erlang flussonic flussonic-transcoder
service flussonic start
```

After installation finishes, activate Flussonic Media Server

ACTIVATING FLUSSONIC MEDIA SERVER

To activate Flussonic Media Server:

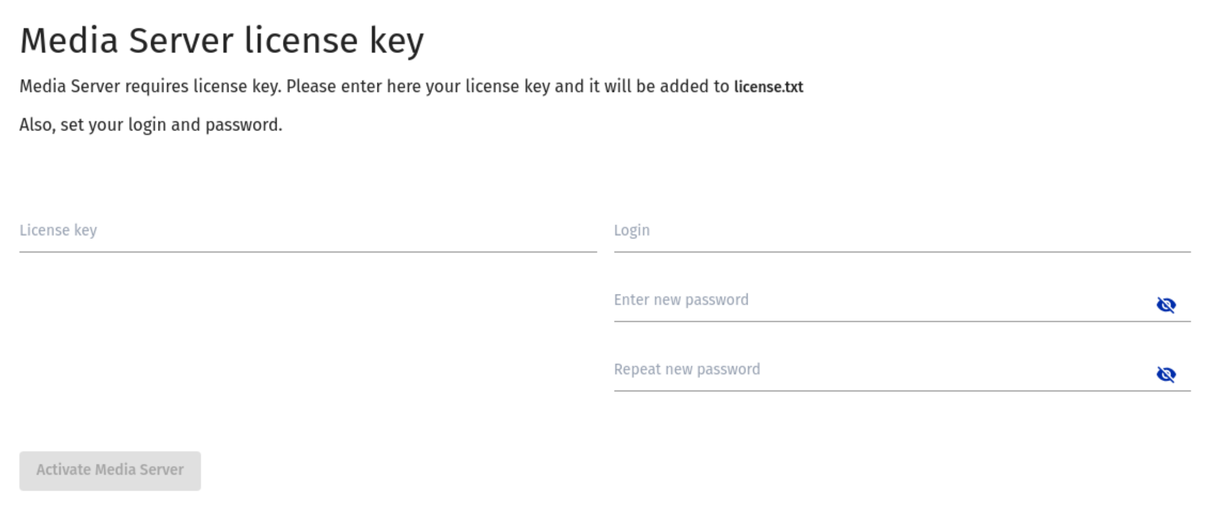
1) Start the server by running the following command in the terminal:

```
service flussonic start
```

2) Open the Flussonic admin UI by entering the `http://FLUSSONIC-IP:80/` URL in the browser. `FLUSSONIC-IP` is the IP address of your Flussonic server.

3) Enter the license key you received and set the administrator's username and password that you are going to use. You can find your license key in the client area on my.flussonic.com/.

> **Warning**
>
> spec_chars_note_en



Read more about Flussonic license at Using the license key.

4) Check if your Flussonic installation is correct by running the following command:

```
service flussonic status
```

Flussonic Media Server is ready to work.

> **Note**
>
> We suggest you do some fine-tuning for best performance with a large number of clients.

> **Warning**
>
> You need to **disable swap** completely as its presence is not compatible with video streaming. If a server doesn't have enough RAM, it can't be extended with `swap` .

About configuration file

When you start the admin UI for the first time and activate Flussonic Media Server, a configuration file is created automatically. It contains the default settings, such as the path to the **Pulse** database and session log.

If you already have an experience with Flussonic, you can also prepare this file manually: specify the login and password in this file and copy it to the server immediately after installation.

HOW TO CHANGE THE ADMINISTRATOR'S PASSWORD?

You can change the administrator's password by editing the config file or using the admin UI.

Editing the configuration file

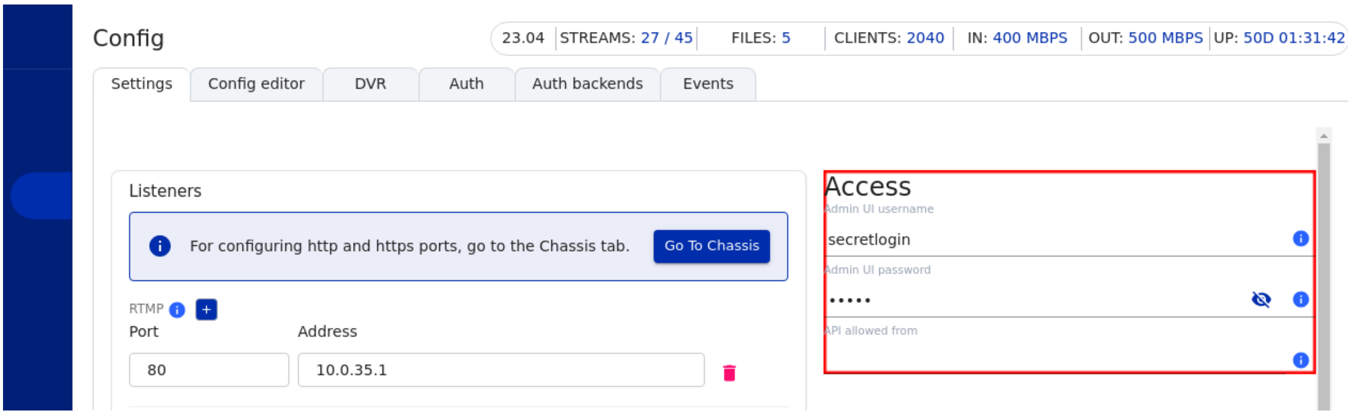
1) Open the `/etc/flussonic/flussonic.conf` config file and change the password in the `edit_auth` directive.

2) To apply the changes, reload the server configuration by running the following command:

```
service flussonic reload
```

Using the admin UI

1) Go to the **Config** page in the navigation menu.
2) Head to the **Settings** tab and find the **Access** section. Enter your new password in the *Admin UI password* field and repeat the password in the field below. Apply the changes by clicking **Save** .



STARTING AND STOPPING FLUSSONIC

To manage Flussonic in the terminal, use the following commands:

- To start the service:

```
service flussonic start
```

- To stop the service:

```
service flussonic stop
```

- To restart the service:

```
service flussonic restart
```

- To reconfigure the service with active client connections:

```
service flussonic reload
```

RUNNING FLUSSONIC IN A DOCKER CONTAINER

Flussonic Media Server is available as a Docker container.

Installing in Docker allows you to run Flussonic on various operating systems that support Docker, not just Ubuntu. Additionally, this installation allows you to take advantage of all Docker benefits: isolation, security, container management, etc.

There are two modes in which you can use Flussonic in Docker. I'll call them the **sysadmin-way** and the **devops-way**.

sysadmin-way

In this configuration, Docker is used as an equivalent to a virtual machine. We recommend this mode only for testing and experiments, small services, and when using only `TCP/HTTP` protocols.

To run Flussonic in a container:

```
docker run -p 8081:80 -v /some/path/flussonic1:/etc/flussonic flussonic/flussonic
```

Or, if you have Nvidia graphics cards that you want to use:

```
docker run --rm -p 8081:80 -v /etc/flussonic:/etc/flussonic --gpus all -e NVIDIA_DRIVER_CAPABILITIES='all' flussonic/flussonic
```

Each container can have its own path for configuration, mapped to a convenient network port. You can access the UI via the port specified in the startup command, create, and edit streams.

UDP capture, QSV, WebRTC may not work, or will work with limitations due to Docker specifics.

devops-way

In this mode, infrastructure parameters are passed through environment variables, and stream configuration is either via `ro`-directory or config-external.

- `STREAMER_HTTP` - allows you to specify the port the container will listen on.
- `STREAMER_EDIT_AUTH` - sets the login-password for access to API and UI.
- `STREAMER_CONFIG_EXTERNAL` - the address of the stream configuration backend.
- `LICENSE_KEY` - license key.

You need to mount the `/etc/flussonic/flussonic.conf.d` directory, from which Flussonic will read all files.

In this mode, the server cannot be configured via API, but you can access the UI for debug purposes, check your streams.

HOW TO CHANGE TIME ZONE ON YOUR SERVER

If you need to change the time zone on the server, this can be done using standard operating system tools. Changing the time zone on the server does not affect the operation of Flussonic Media Server, since all operations internally are carried out in the UTC time zone, including logging. We recommend that you always synchronize your server's time via NTP.

TLS certificate with Let's Encrypt

Let's Encrypt service automatically provides certificates for setting up HTTPS in automatic mode.

Flussonic Media Server has in-built support for Let's Encrypt; installation of extra packages and manual adjustment of a web server is not necessary.

Just open the administrator's interface and click the **Issue by Let's Encrypt** button.

After that *Flussonic Media Server* will automatically retrieve and install the certificate, and you can specify HTTPS port number.

You do not have to worry about certificate expiration date or manually editing text config files.

HTTPS is useful for:

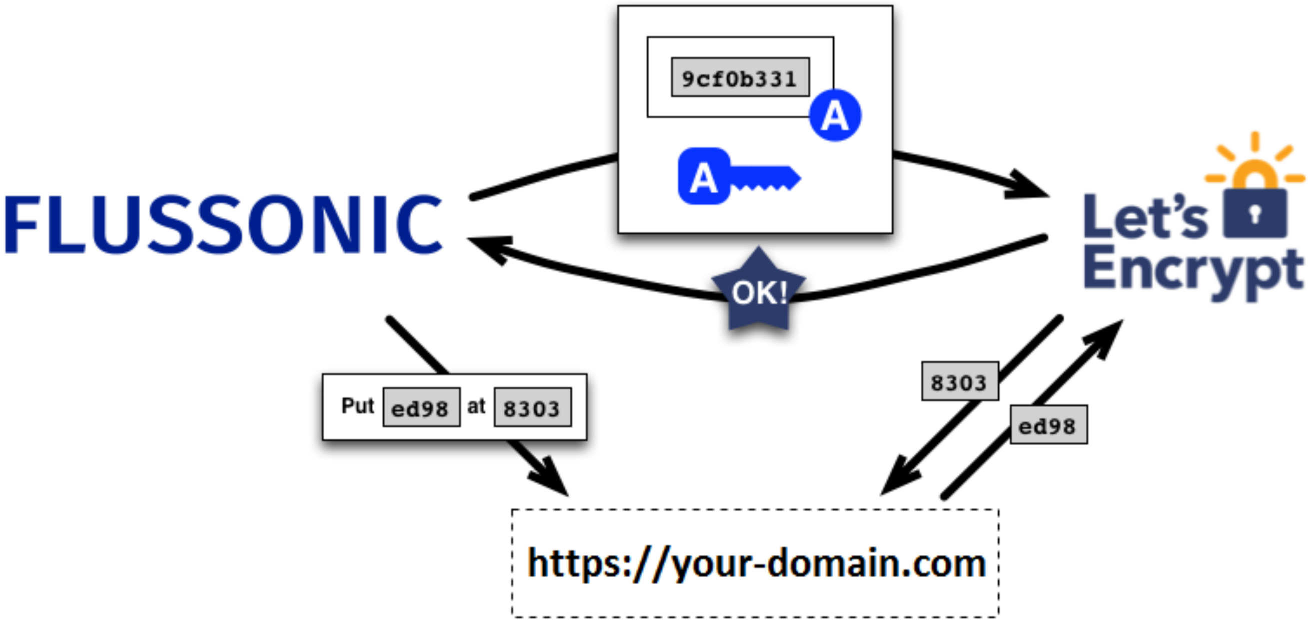
- prevention of server control theft, nobody will be able to intercept your password or streaming links;
- protecting video from security cameras;
- inserting a link to another site running on https (otherwise, browsers will start warning about unprotected content).

Below is more detailed description of the process of setting up, and the operating principle of Let's Encrypt.

LET'S ENCRYPT: HOW IT WORKS

Detailed description can be found on the official site: https://letsencrypt.org/how-it-works/.

To make Let's Encrypt service to issue a valid certificate for you, it is necessary to prove that you own the domain. Click **Issue by Let's Encrypt** in the admin panel. *Flussonic Media Server* will provide the domain name for which a certificate is required. In response, it receives a key that should be returned back when the validating bot will connect to your server via HTTP (exactly on port 80) at address `http://your-domain.com/.well-known`.



The validating bot tries to connect to your domain. The domain must be delegated, and DNS records must be set up for the IP address where *Flussonic Media Server* is operating. The bot verifies your ownership of the domain, and *Flussonic Media Server* saves the certificate.

To extend the certificate, you should repeat the verification process, that means that the *Flussonic Media Server* should always be listening on the port `http 80;`. Verification cannot be done on some other port — this is the rule of Let's Encrypt. The certificate extention occurs automatically when the certificate expires; also, the certificate can be updated manually through the admin panel of *Flussonic Media Server*.

SETTING OF LET'S ENCRYPT CERTIFICATE

1. Open the admin panel of *Flussonic Media Server* using a domain name instead of IP address (e.g., `http://your-domain.com/admin`).
2. Proceed to the **Config** tab.
3. In the **TLS-tunneled protocols** section, click the **Issue by LetsEncrypt** button. This button launches the process of obtaining a certificate.

4. Wait for the certificate expiry date to appear (it usually takes up to 10 seconds).
5. In the **Listeners**, add the port number 443 to the **HTTPS ports** list. You may refer here for details on configuring the listeners.

Save the settings by clicking **Save**. *Flussonic Media Server* will redirect your browser to https:// — now you can provide services over HTTPS.

OBTAINING MULTIDOMAIN LET'S ENCRYPT CERTIFICATE

The procedure described above allows to issue an SSL certificate for only one domain. But what if you run multiple instances of Flussonic (e.g., for delivering streams to multiple TV channels) and need to secure multiple domains with SSL cerificates?

In this case you can use our Let's Encrypt CLI tool that allows to obtain a multidomain certificate. For example, if you have domains `domain1.example.com` and `domain2.example.com` with Flussonic installed, run the following command:

```
/opt/flussonic/contrib/control.erl letsencrypt -d domain1.example.com -d domain2.example.com
```

The Let's Encrypt certificate will be issued for both domains.

Using the license key

Flussonic supports two types of activation:

- Online activation. Online license requires *Flussonic* servers to have Internet access. To activate *Flussonic* online, get the trial online license on the website.
- Offline activation. When you have a closed network without Internet access as for critical infrastructures, such as airports, ships, use a USB license protected by a Guardant USB key. USB license is specific to a version of *Flussonic Media Server*. To receive a USB key or discuss other offline activation options, contact the technical support team and sales team at support@flussonic.com.

In this section:

- How to use the online license key.
- How to use the USB license key.
- What you can do without a license key.

USING THE ONLINE LICENSE KEY

When you first open the web UI *Flussonic* will ask you to enter the license key that you have received after purchasing the license or requesting a trial license.

You can edit the key in two ways:

- in the `/etc/flussonic/license.txt` file
- in the *Flussonic* UI in the **Config** > **License** section

The key looks like so:

`l4|WXHMkfXhFHeNmvDz-M_tb4|r6BzpmVPpjgKpn9IunpFp5lLbCZOp3`

To validate the license key, the server must have access to the Internet via HTTP and HTTPS.

> **Warning**
>
> *Flussonic* licensing server doesn't make any requests to your server, so you don't have to whitelist it when limiting access to your server. For correct *Flussonic* licensing allow outbound connections for HTTP ports 80 and 443 on your server.

Learn more about the activation process on the Activating Flussonic Media Server page.

Binding the license key

Flussonic requires stable connection with the key server to bind the license key.

Migrating the license key to another server

Flussonic validates the license online, so you can move your license to another server if you need to.

To migrate the license key to another server, follow these steps:

1. Shut down *Flussonic* on the first server.
2. Launch *Flussonic* on the new server and enter the license key when you first open the web UI.

USING THE USB LICENSE KEY

To use the USB key, you need the following components that you received:

- USB key
- License key starting with `g4|`
- Activation file

You can also copy or download them in your account.

Use the activation file to run the version of *Flussonic Media Server* that's stated in the file name. If you need to run several different versions, use several different activation files of the corresponding versions.

Activate the USB license after installing *Flussonic* as follows:

1. Without insterting the USB key, run the following command in the terminal:

   `cp /opt/flussonic/contrib/95-grdnt.rules /etc/udev/rules.d/ && service udev restart`

2. Insert the USB key to the server.
3. Start *Flussonic* by running the following command:

   `service flussonic start`

4. Open the *Flussonic* web interface in the browser using the `http://FLUSSONIC-IP:80/` address, where `FLUSSONIC-IP` is the IP address of the *Flussonic* server.
5. On the start page, enter the license key starting with `g4|` and administrator username and password that you are going to use. Then click **Activate Media Server**.

   > **Warning**
   >
   > spec_chars_note_en

6. If the license key is valid, you will see the **Config > Settings** tab in the *Flussonic* Admin UI. Click **Upload Activation file** and select the activation file for the *Flussonic* version installed on the server. You can find the license key and the activation file in the client area on my.flussonic.com.

FLUSSONIC WEB INTERFACE WITHOUT A LICENSE KEY

If the license key is invalid or missing, the *Flussonic* web interface opens in a limited view mode and shows the "no license" page. On this page you can enter the license key or upload the activation file for the USB key.

Without a license key, you will see the **Config > Settings** and **Support** pages. With Flussonic Coder on you will also see the **Chassis** section.

MIGRATING LICENSE KEY TO ANOTHER SERVER

When reinstalling the operating system and Flussonic product on the same server or after updating its hardware components You can reuse your license on the same hardware or after updating its hardware components, because licenses are in no way tied to the hardware/server/kernel/ip address, etc. It is just enough to turn off the 'old' server and turn on the 'new' one. It is needed to delete the license key from the old server and stop flussonic here.

We monitor the number of simultaneously running servers.

When changing a server, calmly start the license on the new one, then remove the license key from the old server and do not forget to turn off the Flussonic service on the old server, so that it does not restart automatically. If you forget, you will have two running servers, and our colleagues will offer you to extend your license. There is no need to notify us in case of a server change.

Updating Flussonic

Update Flussonic Media Server package when a version with new features or with bug fixes is released. Make sure that you update to the latest version of Flussonic every month or at least once every three months. This way, you avoid any unexpected issues with your service.

Our blog and Flussonic admin UI will update you when a new version of Flussonic is available.

> **Warning**
>
> If you have issues with the latest version of Flussonic, please contact support first. If your issue isn't solved, you can roll back to the previous version.

On the page:

- Updating Flussonic in the admin UI
- Updating Flussonic from the terminal on Ubuntu
- Updating Flussonic from the terminal on CentOS
- What version is currently installed
- How to revert to the previous version
- Rolling release updates

UPDATING FLUSSONIC IN THE ADMIN UI

Update your version of Flussonic for the online license key or USB license key.

Updating Flussonic with online license key

1) Install the latest version by clicking **Upgrade** in the sidebar menu of the admin UI.

2) Complete the installation by restarting the service. Click **Restart** and wait till Flussonic restarts.

Updating Flussonic with USB license key

1) Download the activation file from the client area on my.flussonic.com. Go to **License keys** > **Licenses** > your license, select the required version from the list and click **Get Offline Activation File**.

2) Open Flussonic admin UI and go to **Config** > **Settings** and upload the activation file by clicking **Upload Activation File**.

3) Install the latest version by clicking **Upgrade** in the sidebar menu of the admin UI.

4) Complete the installation by restarting the service. Click **Restart** and wait till Flussonic restarts.

UPDATING FLUSSONIC FROM THE TERMINAL ON UBUNTU

```
apt-get update
apt-get -y install flussonic
service flussonic restart
```

> **Warning**
>
> To complete the installation of a new version, make sure you restart Flussonic manually. The last command in the code does it.

UPDATING FLUSSONIC FROM THE TERMINAL ON CENTOS

```
yum -y install flussonic flussonic-erlang flussonic-transcoder
```

The package manager can create the file `/etc/init.d/flussonic.rpmnew`. Rename it like the following:

```
mv /etc/init.d/flussonic.rpmnew /etc/init.d/flussonic
```

Then restart Flussonic:

```
service flussonic restart
```

WHAT VERSION IS CURRENTLY INSTALLED

To check the version of Flussonic Media Server currently installed on a computer, run the following command in the terminal:

```
dpkg -l | grep flussonic
```

HOW TO REVERT TO THE PREVIOUS VERSION

Roll back to the previous version if you have an online license key or a USB license key.

For online license key

> **Warning**
>
> We do not store versions of Flussonic older than nine months.

Specify the version of the `flussonic` package and its dependencies.

1) Get versions of dependencies by using `apt-cache` and specifying the required version of Flussonic:

```
apt-cache show flussonic=24.02  | egrep '^(Depends|Suggests):'
```

The output will be like so:

```
Depends: flussonic-erlang (=26.1.2.3), flussonic-transcoder-base (=23.02.0)
```

2) Install packages with these versions:

> **Danger**
>
> Before installing packages create backups of the configuration files in the directory `/etc/flussonic` and `.db` files in the directory `/opt/flussonic/priv` (this directory is used by default, you can change the path in the configuration file).v

```
apt-get install flussonic=24.02 flussonic-erlang=26.1.2.3 flussonic-transcoder-base=23.02.0
```

> **Danger**
>
> We cannot guarantee that the server operates correctly on those Linux distributions for which we do not provide installation packages.

For USB license key

> **Note**
>
> Activation files are generated for specific license and version of Flussonic.

To return to the previous version of Flussonic, follow the steps in the Updating Flussonic with USB license key.

ROLLING RELEASE UPDATES

New version of Flussonic is released every month. We also have a repository with rolling updates that we release between two major releases. Every day we update it with new Flussonic builds that include new features and bug fixes. Rolling updates are release candidate (RC) versions that we run in our laboratory. We offer release candidate (RC) versions to customers who want to get updates before the next major release comes out.

You can install a rolling update and return back to the major release.

How to install a rolling update

1) Remove the outdated version of Flussonic and its dependencies:

```
apt remove flussonic
```

2) Change the repository to the one with the rolling updates and install Flussonic:

```
mkdir -p /etc/apt/trusted.gpg.d/
curl -L http://apt.flussonic.com/repo/master/dev.key > /etc/apt/trusted.gpg.d/dev.gpg
echo "deb http://apt.flussonic.com/branch/flussonic/master repo/" > /etc/apt/sources.list.d/flussonic.list;
apt update;
apt install flussonic;
service flussonic restart
```

How to return back to the major release

1) Remove the outdated version of Flussonic and its dependencies.

> **Danger**
>
> Before removing the packages, create a backup of the configuration files located in the directory `/etc/flussonic` and the `.db` files in the directory `/opt/flussonic/priv` (this directory is used by default, you can change the path in the configuration file).

```
apt remove flussonic
```

2) Change the repository to the one with official releases and install Flussonic:

```
echo "deb http://apt.flussonic.com binary/" > /etc/apt/sources.list.d/flussonic.list;
apt update;
apt install flussonic;
service flussonic restart
```

> **Danger**
>
> If Flussonic fails to start, run `service flussonic run` and `journalctl -u flussonic -n 100` in the command line terminal and send the output to our technical support team.

Upgrading Flussonic Coder

To get access to new features and fix any bugs, upgrade your Flussonic Coder firmware to the latest version. If you are having issues with Flussonic Coder after upgrading, contact support and roll back to the previous version.

CHECKING FOR NEW VERSIONS

1. Open the Flussonic Coder web interface and go to **Chassis**.
2. In **System Information**, click **Check For New Version**.

In *Firmware Version* drop-down list, you will see new version of the firmware that you can use to upgrade.

UPGRADING AND ROLLING BACK

1. Open the Flussonic Coder web interface and move to the **Chassis** page in the side menu.
2. (For upgrading) Check for new versions by going to **System Information** and clicking **Check For New Version**.
3. In *Firmware Version* drop-down list, select the firmware version and click **Upgrade**. To upgrade to the latest version, click **Upgrade To Latest**.

You will see that the firmware version specified in *Firmware Version* has changed. It means that the upgrade or rollback of the Flussonic Coder was successful.

2.2.2 Maintaining

Fine-tuning Flussonic Media Server and the operating system

This section describes certain common issues and techniques of tweaking the operating system and *Flussonic Media Server* software for working under high load.

UDP CAPTURE SETUP

Flussonic automatically tunes some network setting to optimize the ingest of UDP streams including WebRTC and multicast. If this is not acceptable for you, set the DO_NOT_DO_NET_TUNING environment variable and make the settings manually:

To pass the DO_NOT_DO_NET_TUNING environment variable to Flussonic, execute:

```
systemctl edit flussonic
```

Add the following line to the opened file:

```
[Service]
Environment="DO_NOT_DO_NET_TUNING=true"
```

Save the changes.

Now you'll need to increase the amount of memory allocated to UDP buffers:

```
sysctl -w net.core.rmem_max=1048576
sysctl -w net.core.rmem_default=1048576
sysctl -w net.ipv4.udp_mem="8388608 12582912 16777216"
```

Note that these settings will stay only until system reboot. In order to make those setting persistent open the file `/etc/sysctl.conf` in an editor and add the following lines in the end:

```
net.core.rmem_max = 1048576
net.core.rmem_default=1048576
net.ipv4.udp_mem = 8388608 12582912 16777216
```

To apply the changes, execute the command:

```
sudo sysctl -p
```

WORKING WITH A LARGE AMOUNT OF MEMORY

When more than 60GB of memory is available, we recommend allocating 10GB to the system:

```
sysctl vm.min_free_kbytes=10240000
```

TCP/IP STACK SETUP

If you intend to use *Flussonic Media Server* for broadcasting at more than 3-4 Gbit/s, you might want to fine-tune the system's TCP/IP stack.

First, you will need to allocate more memory to connection buffers:

```
sysctl -w net.core.wmem_max=16777216
sysctl -w net.ipv4.tcp_wmem="4096 4194394 16777216"
sysctl -w net.ipv4.tcp_congestion_control=htcp
sysctl -w net.ipv4.tcp_slow_start_after_idle=0
```

Note that these settings will stay only until system reboot. In order to make those setting persistent open the file `/etc/sysctl.conf` in an editor and add the following lines in the end:

```
net.core.wmem_max = 16777216
net.ipv4.tcp_wmem = 4096 4194394 16777216
```

Then execute the command `sudo sysctl -p` to apply the changes.

You will also need to change the network adapter's settings: `ifconfig eth0 txqueuelen 10000`.

Make sure to check the adapter's driver version. Using the latest version is recommended. Use `ethtool` to find the version of the driver and the firmware:

```
ethtool -i eth2
```

The output will be like:

```
driver: ixgbe
version: 3.15.1
firmware-version: 0x61c10001
bus-info: 0000:04:00.0
```

> **Warning**
>
> If the firmware file in the `/lib/firmware` directory is updated, the server must be rebooted. The old firmware version may remain. Do not forget to run the `update-initramfs` utility before restarting the server.

CONFIGURING NETWORK ADAPTER

Configuring interrupts

Modern 10 Gigabit network adapters support multiple queues for incoming and outgoing packets. Sometimes these queues must be manually linked to different CPU cores.

Without this optimization trick the entire networking subsystem of the server will use only one CPU core. This is how it looks like:

```
cat /proc/interrupts
```

The output will look as follows:

```
           CPU0    CPU1    CPU2    CPU3    CPU4    CPU5    CPU6    CPU7
  0:       2097       0       0       0       0       0       0       0  IR-IO-APIC      timer
...
 66: 2072120005       0       0       0       0       0       0       0  IR-PCI-MSI      eth2-TxRx-0
 67:    1562779       0       0       0       0       0       0       0  IR-PCI-MSI      eth2-TxRx-1
 68:    1830725       0       0       0       0       0       0       0  IR-PCI-MSI      eth2-TxRx-2
 69:    1504396       0       0       0       0       0       0       0  IR-PCI-MSI      eth2-TxRx-3
 70:    5112538       0       0       0       0       0       0       0  IR-PCI-MSI      eth2-TxRx-4
 71:    2229416       0       0       0       0       0       0       0  IR-PCI-MSI      eth2-TxRx-5
 72:    1686551       0       0       0       0       0       0       0  IR-PCI-MSI      eth2-TxRx-6
 73:    1217916       0       0       0       0       0       0       0  IR-PCI-MSI      eth2-TxRx-7
 74:       2358       0       0       0       0       0       0       0  IR-PCI-MSI      eth2
```

For Intel adapters, the manufacturer provides the `set_irq_affinity` script, which distributes the queues to different cores. After running the script, the interrupts data looks like this:

```
           CPU0       CPU1       CPU2       CPU3       CPU4       CPU5       CPU6       CPU7
  0:       2097          0          0          0          0          0          0          0  IR-IO-APIC  timer
...
 66: 2072120005          0          0          0          0          0          0          0  IR-PCI-MSI  eth2-TxRx-0
 67:    1562779 1162738082          0          0          0          0          0          0  IR-PCI-MSI  eth2-TxRx-1
 68:    1830725          0 1133908105          0          0          0          0          0  IR-PCI-MSI  eth2-TxRx-2
 69:    1504396          0     177620 1123678951          0          0          0          0  IR-PCI-MSI  eth2-TxRx-3
 70:    5112538          0          0          0 1638450740          0          0          0  IR-PCI-MSI  eth2-TxRx-4
 71:    2229416     130189          0          0          0 1441511712          0          0  IR-PCI-MSI  eth2-TxRx-5
 72:    1686551          0          0          0          0          0 1402472725          0  IR-PCI-MSI  eth2-TxRx-6
 73:    1217916          0          0      66145          0          0          0 1380402032  IR-PCI-MSI  eth2-TxRx-7
 74:       2358          0          0          0          0          0          0          0  IR-PCI-MSI  eth2
```

This setting becomes critical when the traffic reaches the vicinity of 3-5 Gbit/s.

Configuring the connection to a switch

If you connect server network adapter to a switch, please check that both sides have compatible settings. You should either use *auto select* settings on both sides, or strictly the same speed and duplex.

OPTIMIZING THE SERVER FOR VOD

Optimization of the server for Video On Demand is discussed in detail in a dedicated section.

SWITCHING THE CPU TO THE HIGH PERFORMANCE MODE

In Linux the `scaling_governor` knob is in power save mode by default. In this case, the server does not use all of its hardware resources. For the server to work in high performance mode, do the following:

Disable the `ondemand` controller:

```
systemctl disable ondemand
```

Reboot the server:

```
reboot
```

Check the current value of `scaling_governor`:

```
cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
```

Flussonic Media Server migration

When migrating Flussonic Media Server from one server to another, you should copy the config and license files. It is also possible to transfer the archive.

> **Warning**
>
> Do not move executable files and installed libraries. Use the package manager to install Flussonic Media Server on the new server.

MIGRATING THE CONFIG ANG LICENSE

To migrate the configuration:

1. Install Flussonic Media Server on the new server.

```
curl -sSf https://flussonic.com/public/install.sh | sh
```

1. Move the following files from the old server to the new one using one of the methods described later on this page:
    - `/etc/flussonic/flussonic.conf` — the main configuration file.
    - `/etc/flussonic/license.txt` — license.
2. Stop Flussonic Media Server on the old server:

```
service flussonic stop
```

1. Start on the new server:

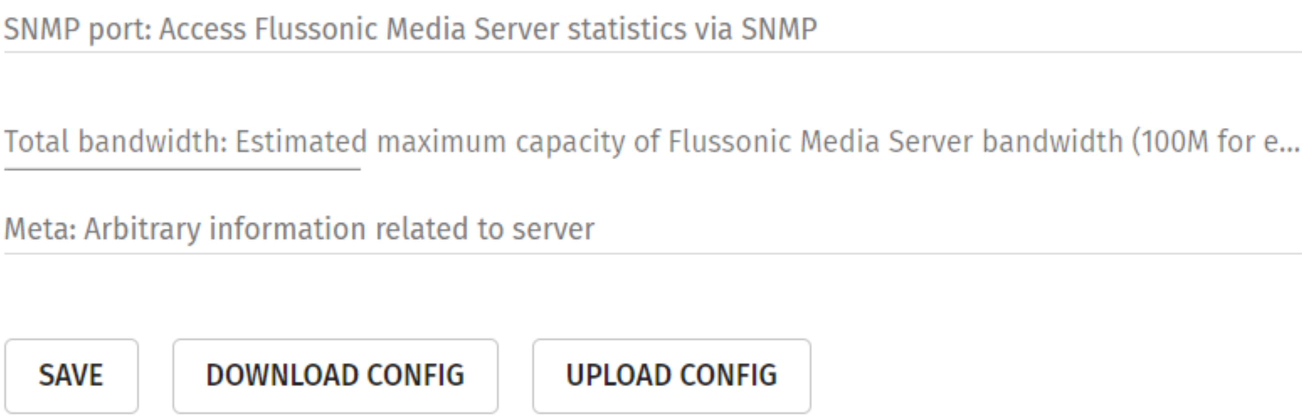
```
service flussonic start
```

Ways to transfer files:

- Transferring the configuration using web interface
- Transferring the configuration using SCP
- Transferring a Configuration Using USB Media

Transferring the configuration using web interface

Go to the **Config** -> **Settings** tab both on the new and the old server. To download the configuration file, click the **Download Config** button. To upload configuration file to the new server, click the **Upload Config** button.



Your license key can be viewed in the client area on the **License keys** tab.

Transferring the configuration using SCP

SCP (Secure CoPy) is a program for transferring files over a network between hosts. It uses SSH for data transfer, including authentication and security protocols that are implemented in SSH.

To copy the configuration and license files from one remote server `remote.host1` to another remote server `remote.host2`, execute the following command:

```
scp user@remote.host1:/etc/flussonic/flussonic.conf user@remote.host2:/etc/flussonic/
scp user@remote.host1:/etc/flussonic/license.txt user@remote.host2:/etc/flussonic/
```

Transferring configuration using USB media

If you want to transfer configuration files using any USB media, use the following instruction.

Mounting USB

First, create the mount point (directory):

```
mkdir -p /mnt/usb
```

Insert the USB flash drive into the USB port and find the name of the attached device:

```
fdisk -l
```

The result of this command will look like:

```
Disk /dev/sdb: 4008 MB, 4008706048 bytes
118 heads, 53 sectors/track, 1251 cylinders, total 7829504 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x74a37a4d

Device    Boot      Start         End      Blocks   Id  System
/dev/sdb1   *          63     7829503     3914720+   b  W95 FAT32
```

In this example the device name is `/dev/sdb1`.

Use it to mount the device:

```
mount /dev/sdb1 /mnt/usb
```

Copying the configuration

```
cp /etc/flussonic/flussonic.conf /mnt/usb/flussonic.conf
cp /etc/flussonic/license.txt /mnt/usb/license.txt
```

After copying, do not forget to unmount the drive:

```
sudo umount /dev/sdb1
```

Installing the configuration on a new server

Install *Flussonic Media Server* on the new server:

```
curl -sSf https://flussonic.com/public/install.sh | sh
```

Create a directory in which the USB-drive will be mounted:

```
mkdir -p /mnt/usb
```

Insert the media into the USB port and find the name of the device:

```
fdisk -l
```

Mount:

```
mount /dev/sdb1 /mnt/usb
```

Transfer the configuration files:

```
cp /mnt/usb/flussonic.conf /etc/flussonic/flussonic.conf
cp /mnt/usb/license.txt /etc/flussonic/license.txt
```

Launch *Flussonic Media Server*:

```
service flussonic start
```

Done!

MIGRATING ARCHIVE

Use the replication mechanism to migrate the archive.

Archive migration is only possible in a normal operation mode, that is, when the old server continues to work simultaneously with the new one for some time. Note that you will need a license allowing at least one more server for that. For example, you cannot run multiple instances of Flussonic Media Server at the same time if your license key is valid for only one server.

If the migration occurs due to a malfunction of the old server, then the archive will not be transferred. Just start recording the archive again on the new server. To avoid losing the archive in the future, set up replication in advance.

Securing Flussonic

In this section you will learn how to limit access to the *Flussonic* Administration panel and the server.

> **Danger**
>
> If hackers get access to your *Flussonic Administration UI*, they will be able to read and modify any file on the disk.

LOGIN AND PASSWORD

Flussonic allows you to set two types of access in config: `view_auth` and `edit_auth`.

- `view_auth user password;` is used for access to readonly API *Flussonic* functions:

`getting streams info`, `status` and `statistics`.

- `edit_auth user password;` is used for giving the full access to *Flussonic*.

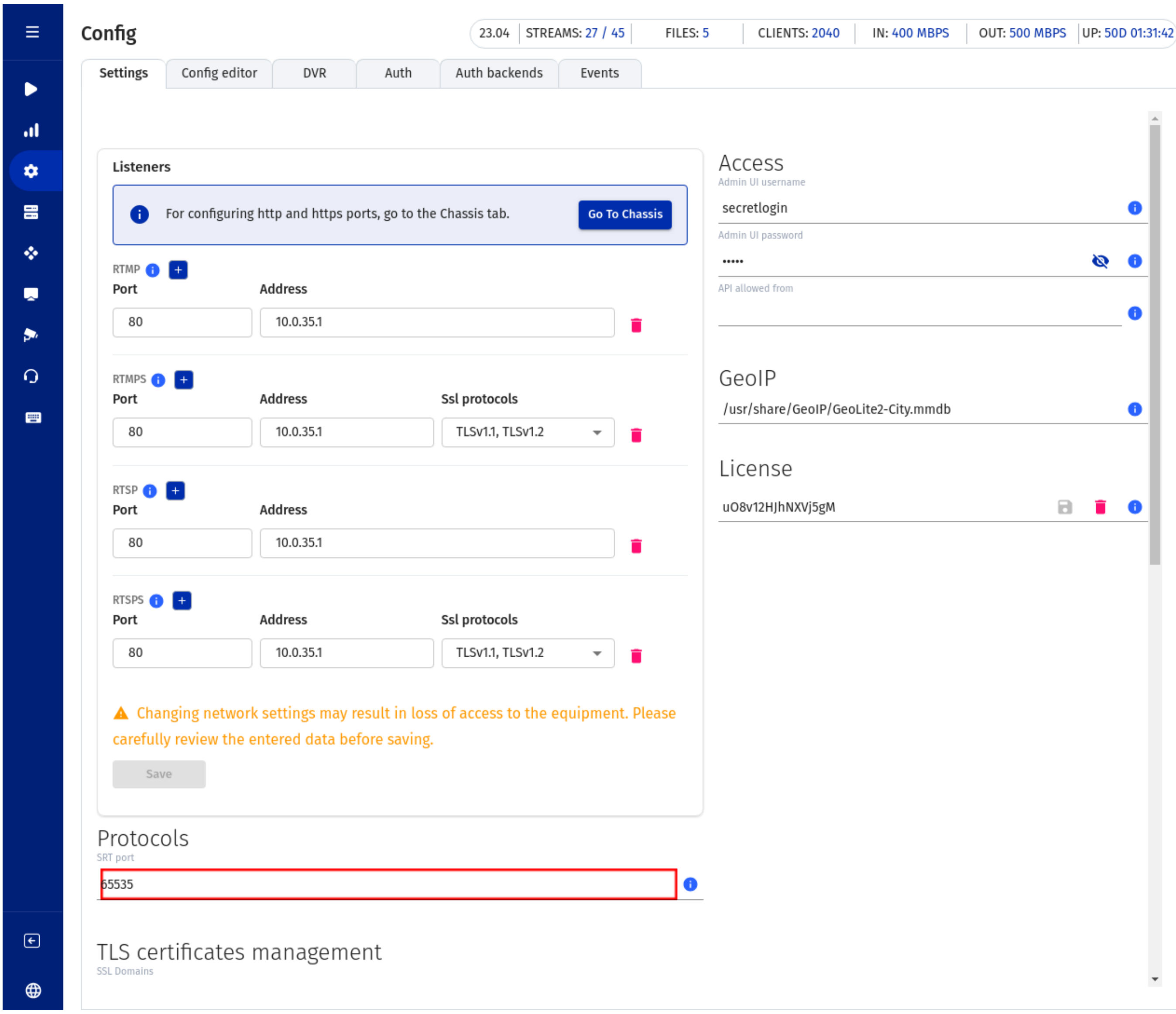
Flussonic can store the password in a hashed format. We know that clients often forget their passwords and look them up in the config file, even when is is not stolen.

We recommend using this option if the server is used by a group of people and the company uses automated tools to track passwords that are stored unprotected.

LIMITING ACCESS TO FLUSSONIC UI BY IP ADDRESSES OR PORTS

Access to *Flussonic UI* is restricted on "white list" basis, i.e. you should list all the ports where you want *Flussonic UI* available. For that, go to the **Listeners** section of the **Config** tab and specify ports for *Flussonic* to listen. You may leave the **Address** empty to allow all IP addresses or specify a value to allow requests by the specified IP only.

Optionally, for HTTPS, RTMPS, and RTSPS, you can select the version(s) of **SSL protocols** you wish to use on the specified IP and port. Clients not supporting these versions will be denied.



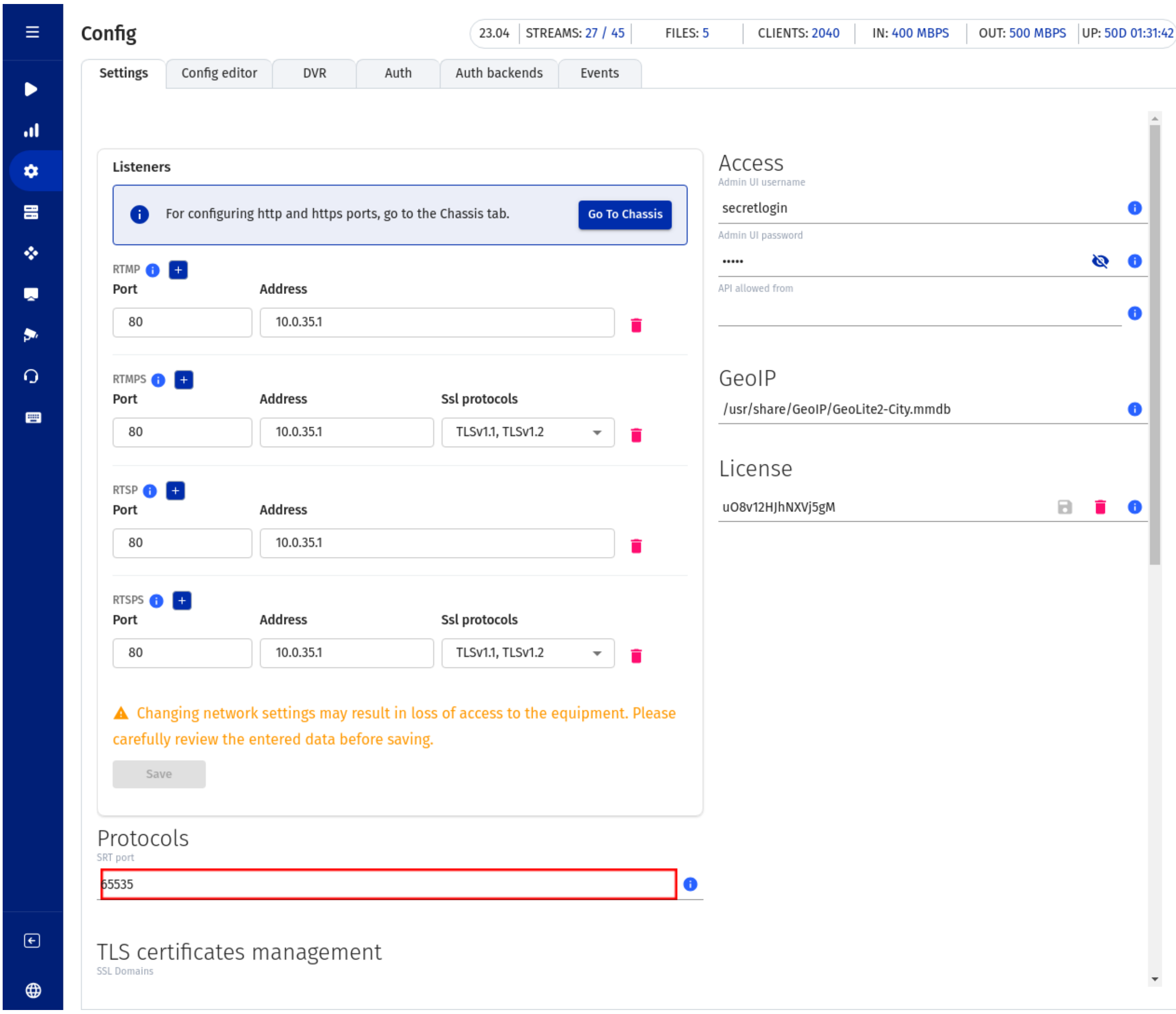
> **Warning**
>
> Be careful when changing the **Address** setting. Incorrect configuration may lead to *Flussonic UI* becoming unavailable from your computer, for example if you specify an IP address in local network while making configurations through the Internet. Make sure you have an alternative way to access the server in case of loss of access to UI, e.g. it is available physically or by SSH.

LIMITING API CALLS BY IP ADDRESSES OR PORTS

By default, *Flussonic* processes API requests on all HTTP or HTTPS ports you specify in config.

You can configure listeners in *Flussonic* to forbid API calls on the specified port(s) in one of the following ways:

- in *Flussonic UI*, go to the **Listeners** section of the **Config** tab to disable the **API** switch for IP addresses and ports where you do not want to accept API calls:



- in the configuration file (`/etc/flussonic/flussonic.conf`), add `api false;` directive in the parameters of the IP address and port you wish to forbid for API calls:

```
https 443 {
    api false;
}
```

UPLOADING SSL CERTIFICATES

If you already have an SSL certificate for *Flussonic* issued by a third-party provider or generated by yourself, you can upload it from your computer to the server through the *Flussonic*'s web interface.

1. First, specify the port for HTTPS. Open the UI and go to **Config** -> **Settings** -> **Listeners** and enter the port for HTTPS, for example, 443.
2. Then go to the **TLS-tunneled protocols** and click **Upload certificates**, choosing the certificate and the key files. Also the CA-certificate may be uploaded.

If you encounter an error when trying to upload your certificate, check that you have only one certificate in the file. As of now, Flussonic's UI does not support uploading files with anything (like root and/or intermediate certificate, key, etc.) except the certificate in them. Please use other means to add such certificates, for example send them over SSH.

Any SSL certificates used by *Flussonic* are stored in a single folder — `/etc/flussonic` or `/etc/streamer` (in a cluster installation). *Flussonic* will automatically rename files to `streamer.crt`, `streamer-ca.crt`, and `streamer.key`.

To remove the uploaded files related to a certificate, click a recycle bin icon in **Config** -> **TLS-tunneled protocols** next to the file list.

GENERATING SSL CERTIFICATES

In order to switch the Administrator's web interface to HTTPS, you need to enable the port for HTTPS in the *Flussonic* configuration. Open the web interface and specify the port for HTTPS in **Config** -> **Settings** -> **Listeners**, for example, 443.

You can generate your own SSL certificate. Below are commands that you should run one by one to generate a *Flussonic*'s own self-signed certificate. Each time the system prompts you to enter the password for the certificate, press **Enter** without typing anything.

```
cd /etc/flussonic

openssl genrsa -des3 -out streamer.key 1024

openssl req -new -key streamer.key -out streamer.csr

mv streamer.key streamer.key.org

openssl rsa -in streamer.key.org -out streamer.key

openssl x509 -req -days 365 -in streamer.csr -signkey streamer.key -out streamer.crt
```

Then put the resulting files to `/etc/flussonic` (`/etc/flussonic/streamer.crt` and `/etc/flussonic/streamer.key`). Alternatively, you can upload these files through the web interface. To do this, go to **Config** > **SSL-tunneled protocols** and click **Upload certificates**.

Intermediate and CA certificates will be taken from `/etc/flussonic/streamer.crt`.

For the most recent OpenSSL commands description, refer to the manual pages in the OpenSSL documentation.

LET'S ENCRYPT CERTIFICATES

Let's Encrypt is offering free SSL certificates with 1-month expiration since April 2016. The certificate is issued in automatic mode.

We have added the support for LetsEncrypt into *Flussonic*. How to setup LetsEncrypt

PROTECTING CONFIGURATION FILE

You can prevent the configuration file from being modified via the API (web interface). JFor this you should create the file `/etc/flussonic/flussonic.conf.locked` by executing the following command:

```
touch /etc/flussonic/flussonic.conf.locked
```

With this file in place nobody will be able to change *Flussonic* settings via the web UI.

RUNNING FLUSSONIC AS AN UNPRIVILEGED USER

You can run *Flussonic* as an unprivileged user. Run the following commands:

```
adduser flussonic --home /var/lib/flussonic --disabled-password
chown -R flussonic /etc/flussonic/
chown -R flussonic /var/lib/flussonic/
chown -R flussonic /var/run/flussonic /var/log/flussonic /etc/flussonic/.erlang.cookie
setcap cap_net_bind_service=+ep /opt/flussonic/lib/erlang/erts-*/bin/x86_64-linux-gnu/beam.smp
```

Then create override systemd unit using `systemctl edit flussonic` command:

```
[Service]
User=flussonic
Group=flussonic
```

To make *Flussonic* run as 'root' again, empty override file.

PROTECTING VIDEO FROM VIEWING BY THE ADMINISTRATOR

By default, the users with *Flussonic* Administrator rights can play back any stream by using the Administration UI. The special Administrator's authorization token is used for that.

You may want to prohibit viewing some streams by the Administrator - streams protected by authorization.

To prevent the *Flussonic* Administrator from playing back any stream that needs authorization:

1) Edit *Flussonic* service unit file (`/lib/systemd/system/flussonic.service`) - do it by using the systemd's override mechanism.

```
systemctl edit flussonic
```

This command opens a text editor (`nano` by default).

2) Add these lines:

```
[Service]
Environment=STREAMER_ADMIN_VIEW_DISABLE=true
```

Press `Ctrl-X`, then `Y` to save and exit.

3) Restart *Flussonic*:

```
service flussonic restart
```

Now if a stream requires authorization, the player in the *Flussonic UI* will return a 403 error at the attempts to play the stream back with an Administrator's token.

Streams without configured authorization will be played back as usual.

PROTECTING THE FILE SYSTEM FROM ACCESS VIA THE UI

In the *Flussonic UI*, the user (Administrator) sets paths to VOD, DVR, and cache. You can configure *Flussonic* to limit the user to certain directories, so that *Flussonic* will allow storing files only in that directories and subdirectories. For example, this allows you to protect the `/root` directory.

Flussonic checks the paths in `vod vod`, `dvr`, `cache`, `copy`, and in the schemas `playlist:///` and `sqlite:///`.

To configure this, add the environment variable `FLUSSONIC_DATAPATH` and specify the uppermost directory allowed for creating VOD, DVR, cache and so on.

> **Warning**
>
> In order for *Flussonic* to restart successfully with the new settings, make sure the current configuration does not have paths to the directories located above the one specified in the `FLUSSONIC_DATAPATH` variable.

To add `FLUSSONIC_DATAPATH`, you can use the systemd's override mechanism:

```
systemctl edit flussonic
```

This command opens a text editor (`nano` by default). Add directories in the following way:

```
[Service]
Environment=FLUSSONIC_DATAPATH=/storage:/mount:/copy
```

Press `Ctrl-X`, then `Y` and `Enter` to save and exit.

Restart *Flussonic*:

```
service flussonic restart
```

Users will be limited to `/storage`, `/mount` and `/copy` and their subdirectories.

Flussonic and firewall

> **Warning**
>
> **Don't** install other software along with Flussonic on the server. Multiple software can conflict with each other and with Flussonic, and its integration isn't covered by the basic support.

Firewall is software to protect the network from unauthorized access based on a defined set of rules. A firewall can block both inbound and outbound connections.

In video streaming, an enabled firewall doesn't protect the server but causes issues. On the video streaming server, only those ports are open that are required to serve clients, and the firewall closes access to the server by blocking ports and doesn't analyze the traffic itself. If you want to improve security, remove the port from the public interface.

When you install Flussonic on a server and specify the admin port, you have two open ports (three with HTTPS port 443): HTTP port 80 and SSH port. A firewall has nothing to protect.

If you can't do without a firewall because of paper security and compliance, consider the following:

- It's important to distinguish rules on inbound and outbound connections. Since there are two ports open on the Flussonic server, it makes no sense to restrict inbound connections and outbound connections. If a malicious hacker has connected to the server, it's too late to protect the server.
- The firewall affects WebRTC streaming as Flussonic selects a random port for WebRTC. For the same reason, the firewall can also affect UDP (User Datagram Protocol) multicast data transmission.
- The firewall blocking outgoing connections restricts access to the licensing system. Flussonic initiates the connection to the licensing server. If you block outgoing connections, you lose access to the license. The question about which IP address to allow for the license to work isn't correct, because it's not clear whether to establish inbound or outbound connections. The IP address of the licensing server isn't static and can change over time. So if you add it to your firewall rules, it's temporary.
- When you contact technical support, engineers will ask you to disable the firewall. In 90% of cases disabling the firewall solves the issue.

To learn about server security, see Securing Flussonic.

Support

Find help and open a support ticket for Flussonic Media Server.
Flussonic provides global technical, pre-sales, billing, and subscription support for Flussonic Media Server and Flussonic Watcher products. Support is available via email and ticketing system for Flussonic paid and trial subscriptions.

FIND HELP WITHOUT OPENING A SUPPORT TICKET

Before creating a support ticket, check out the the technical documentation at https://flussonic.com/doc/ for content such as how-to information or configuration samples for IT professionals and developers.

OPEN A SUPPORT TICKET

If you are unable to find answers by using self-help resources, we encourage you to open an online support ticket. You should open each support ticket for only a single problem, so that we can connect you to the support engineers who are subject matter experts for your problem. Also, Flussonic engineering team prioritize its work based on incidents that are generated, so you're often contributing to service improvements.

The online chat and Flussonic web-forum are not official channels for technical support. They can be used for a quick consultation only.

Support tickets with detailed descriptions of issues will have the priority when we handle issues.

A support ticket can be opened on your User Account Page. Alternatively, you can open a support ticket by sending email to support@flussonic.com.

What information should I include in my support ticket?

In case when you are having an issue with Flussonic software, you can take the following steps and provide us with the below details, so that we can quickly help you resolve the issue:

- Select **Config** in Flussonic main menu, scroll down to **Additional** and set the log level **Debug**; don't forget to save the settings.
- Try to reproduce the issue or wait for its repetition. Thus, information will appear in the log file.
- Select **Support** in Flussonic main menu. To upload debug information through the Watcher UI, go to the **Health** page and click the **Upload debug info** button.
- Write a detailed description of the issue that you need our help with. Please avoid using vague phrases like "it's not working". We are looking for some explanation of what you have expected to happen, and what happened instead. We also ask you to provide stream names, device information (operating system, browser version or set-top box model) and other important information that is always needed by the support team.
- After debug data is uploaded, the system will display upload UUID string on the screen. **Please send the UUID string to us**, for we need it to identify your log files.
- We ask you to not send logs in the Microsoft Word format - those will be deleted.
- If Flussonic server would not start try to launch it manually using the command **service flussonic run**, and then capture the output on the console screen. Copy the contents of the console and send it to us as a **.txt** file, please do not send us screenshots.

Uploading debug from the console

If you cannot open the UI for some reason, you can upload the debug information by the following command:

```
service flussonic upload-logs
```

When the command completes you will receive the UUID that you should send us.

PROVIDING SSH ACCESS TO YOUR SERVER

In some cases, our support team will ask you to provide the root SSH access to your server. This is needed, for example, when support engineer is looking for memory leaks, repairing damaged archive files on the hard disk, solving problems with UDP sources, etc.

To provide access please add **our public key** to the file `/root/.ssh/authorized_keys` in the root user directory.

The key can be added using this **shell script**. You can download and execute the script with the `root` user rights using the following commands:

```
sudo -i
curl -s https://flussonic.com/public/ssh-access.sh | sh
```

After Flussonic public SSH key is added to your system, please provide us with the IP address of your server. We suggest that you configure the SSH port in your system so that it differs from the standard one (22).

We will let you know when the work is done and you can remove our key to revoke access to your system.

Alternative way to provide us with SSH access is to use the button **Enable SSH Access** on the **Support** page of the Flussonic user interface. When you click this button, the system will automatically add our public SSH key to your system and establish SSH tunnel to our support servers.

> **Warning**
>
> Do not send us a plain password for SSH access. It's insecure. We don't provide support using remote access software, such as AnyDesk, TeamViewer, or VNC. We require SSH connection to access your system for troubleshooting. We won't be able to provide you with public IP addresses that will be used to access your server.

Troubleshooting tools

For some troubleshooting tasks we use `screen` and `tcpdump` utilities. If those tools are not installed in your system, please install them with this command:

```
apt-get -y install screen tcpdump
```

LOGS

The single important source of information for error diagnostics and troubleshooting in Flussonic Media Server is log files. By default, Flussonic logs are being written to the directory `/var/log/flussonic`.

The system writes logs into `flussonic.log` file. When the size of this file reaches 40 MB, the rotation is performed:

- The system archives the original file into `flussonic.log.1.gz` and then continues logging into `flussonic.log`.
- The new log file is archived into `flussonic.log.1.gz` and the previous `flussonic.log.1.gz` archive is renamed into `flussonic.log.2.gz`, and so on. The system stores up to 40 such archives.

> **Note**
>
> Such rotation is also applied to other, more specific types of log files (`crash.log`, `access.log`, and so on).

In case Flussonic Media Server does not generate log files, or if the system would not start, please try to launch Flussonic in `foreground` mode and capture the messages in the system console. Use the following command to launch flussonic:

```
service flussonic run
```

Often the root causes of issues with Flussonic Media Server lie in other problems in your system. Please examine and share with our support engineers the log files `/var/log/kern.log` and `/var/log/syslog`.

Log records are done in the UTC time zone and Flussonic offers no way to change this. This approach might be inconvenient if you use only one time zone, but it's the only really good way to deal with things such as daylight saving time, or maintaining and giving technical support for servers located in different time zones.

Stream configuration templates

With an increasing number of streams with similar settings (10+), it becomes a challenging task for an Administrator to manage. Keeping an eye on every stream and adjusting the settings of every stream is ineffective and time-consuming. It also increases the likelihood of making a mistake or missing something. That is where *stream configuration templates* or just *templates* come in handy.

Stream configuration template

defines a set of settings to be applied to several streams to provide a more organized and manageable way of configuration for the streams.

The advantages of using templates are:

1. *Configuration pieces reusability.*
2. *Decomposing complex configurations into simpler and more manageable pieces.*
3. *Decreasing duplication of configuration settings.*
4. *Simplifying change management to duplicate pieces of settings for different streams.*
5. *Increasing clarity and readability of the configuration.*
6. *Reducing the amount of time and effort to manage configuration settings and keeping it up-to-date.*

All in all, stream configuration templates in *Flussonic* help you manage the settings of many streams.

FLUSSONIC CONFIGURATION FILE

The *Flussonic*'s entire configuration is stored in a single file — `/etc/flussonic/flussonic.conf`. The configuration has its own format, it is not JSON, YAML, or INI, but it is very simple and easy to read, which is why *Flussonic* Administrators often open the file and read it. Reading this file is faster than opening several pages in the web interface; configurations of a dozen streams are shown on a single screen of a text editor, and all global settings are immediately visible.

The simple syntax of the configuration file makes it convenient for editing too. Advanced *Flussonic* users often write the configuration in a text editor, the same way as they do when working with other server software, such as web servers.

```
http 80;
edit_auth flussonic password;

stream example {
    input udp://192.168.0.1:5000;
    dvr /storage 7d;
}
```

This is an example of a real configuration, where six lines are enough to define the port that Flussonic will listen on, set a password for the web interface, create a stream and configure its recording.

TEMPLATES

We decided to make the configuration of many streams more convenient. In *Flussonic 21.03* we introduced configuration templates, the `template` section and the `template` option. This is what the same example looks like now:

```
template t1 {
    transcoder vb=1000k deinterlace=true ab=128k;
    dvr /storage 1d;
}
stream channel1 {
    input udp://239.255.0.1:1234;
    template t1;
}
stream channel2 {
    input udp://239.255.0.2:1234;
    template t1;
}
```

All general settings of streams are placed in a separate section, and only unique settings are defined within a stream. If there are at least 10 streams, you can already see how much more compact `flussonic.conf` becomes.

What's more, it is enough to assign the template to a stream once, and then work only with its configuration. This way, synchronization of stream settings on a cluster of transcoders will be reduced to copying the `template` between servers.

The `template` section supports the same options as the `stream` section.

SETTINGS OVERRIDING IN STREAM CONFIGURATION

If one of your streams needs to override any of the parameters defined in the template, this can be done as follows:

```
template t1 {
    transcoder vb=1000k deinterlace=true ab=128k;
    dvr /storage 1d;
}
stream channel1 {
    input udp://239.255.0.1:1234;
    template t1;
    dvr s3://minioadmin:minioadmin@minio:9001/test 3d;
}
```

The local configuration of the stream `channel1` has a priority over the setting from `template t1`. We recommend using this for testing or in rare cases because otherwise the templates will lose their purpose, a large number of overrides will return you to the situation when you had to manually track the configuration of each stream.

GLOBAL OPTIONS OF STREAMS

Options such as `on_play`, `url_prefix`, `cluster_key` could have been specified for all streams at once, but then there was a problem with exceptions control.

```
on_play http://middleware_example/auth;

stream channel1 { # the stream uses global auth
    input udp://239.255.0.1:1234;
}
stream channel2 { # the stream overrides global auth with local defined
    input udp://239.255.0.2:1234;
    on_play securetoken://key;
}
```

In practice, it often turned out that not all streams needed to inherit the general configuration, and Administrators refused to use it. Explicit definition is clearer, better readable, and less error prone than implicit inheritance.

```
stream channel1 {
    input udp://239.255.0.1:1234;
    on_play http://middleware_example/auth;
}
stream channel2 {
    input udp://239.255.0.2:1234;
    on_play securetoken://key;
}
```

Global options are very similar to templates, right? Therefore, starting from 21.03 you will see the message:

```
# Stream templates:
# Template globals currently applies to all streams without templates.
# This will change in future, explicit template usage is recommended.
#template globals {
    on_play http://middleware_example/auth;
#}
```

In this release we will leave this unchanged but we plan to move the configuration to a separate template soon, which will be used by all streams without the specified `template`.

It becomes clear now that a stream can inherit configuration from only one template: either an explicitly specified one or global one, but not from both.

TEMPLATES AND PREFIXES

This option is connected with dynamic names for the streams. Template creates one publishing point with one or more publishing locations, depending on the number of prefixes you define.

Dynamic name is a term used to describe a name of a publishing stream not known to *Flussonic* in advance.

`prefix` is used to form a stream name. The general structure for a stream name is as follows: `PREFIX/STREAM_NAME`.

So the configuration may look like this (`input publish://` is crucial here):

```
template example_template {
  prefix foo;
  prefix bar;

  input publish://;
  backup priv/bunny.mp4;
  source_timeout 2;
}
```

We specified two prefixes in `example_template` : `foo` and `bar` , which enabled a backup file and a source timeout.
So when you publish the stream to *Flussonic*, the name of the stream will have one of the two possible formats, depending on chosen publishing location: `foo/STREAM_NAME` or `bar/STREAM_NAME` .
For example, if you publish an RTMP stream to `foo` , the URL will look as follows: `rtmp://FLUSSONIC-IP/foo/STREAM_NAME` .

Simply put, all settings within the template with prefixes apply to the streams published under the name of the prefix(es) (`foo/STREAM_NAME` and `bar/STREAM_NAME` in our example).

It is possible to use a special empty prefix (`""`). In this case the template allows to publish a stream with any prefix or even without a prefix.

2.2.3 Monitoring

SNMP

Flussonic Media Server has a basic implementation of the SNMP protocol. It allows monitoring of various parameters such as resource consumption by *Flussonic*'s video streams.

To use it, add the following lines in the *Flussonic* configuration file:

```
snmp 4000;
```

Apply the settings:

```
service flussonic reload
```

This will enable a listener for SNMP on port 4000.

snmpwalk

To fetch stats via SNMP, run the following commands:

```
apt-get -y install snmp snmp-mibs-downloader

snmpwalk -c USERNAME -v 2c -M +/opt/flussonic/lib/mibs -m +STREAMER-MIB 127.0.0.1:4000 .
```

Replace `USERNAME` with the login of the *Flussonic* Administrator.

Here `snmpwalk` is a utility for diagnosing an installed SNMP system.

The option `-c USERNAME` means "community" in terms of SNMP. SNMP community is equal to the *Flussonic* Administrator's login.

Example

If everything is configured correctly, the response of the `snmpwalk` utility will look like the following:

```
# snmpwalk -c flussonic -v 2c -M +/opt/flussonic/lib/mibs/ -m +STREAMER-MIB 127.0.0.1:4000 .
SNMPv2-SMI::mib-2.1.1.0 = STRING: "Streamer 21.04"
SNMPv2-SMI::mib-2.1.2.0 = OID: STREAMER-MIB::streamerModule
SNMPv2-SMI::mib-2.1.3.0 = Timeticks: (668596) 1:51:25.96
SNMPv2-SMI::mib-2.1.4.0 = STRING: "support@flussonic.com"
SNMPv2-SMI::mib-2.1.5.0 = STRING: "Streamer"
SNMPv2-SMI::mib-2.1.6.0 = STRING: "Erlang"
SNMPv2-SMI::mib-2.1.7.0 = INTEGER: 72
SNMPv2-SMI::mib-2.1.8.0 = Timeticks: (0) 0:00:00.00
SNMPv2-SMI::mib-2.11.1.0 = Counter32: 9
SNMPv2-SMI::mib-2.11.3.0 = Counter32: 0
SNMPv2-SMI::mib-2.11.4.0 = Counter32: 0
SNMPv2-SMI::mib-2.11.5.0 = Counter32: 0
SNMPv2-SMI::mib-2.11.6.0 = Counter32: 0
SNMPv2-SMI::mib-2.11.30.0 = INTEGER: 1
SNMPv2-SMI::mib-2.11.31.0 = Counter32: 0
SNMPv2-SMI::mib-2.11.32.0 = Counter32: 0
STREAMER-MIB::streamsNum.0 = Gauge32: 3
STREAMER-MIB::sIndex.1 = INTEGER: 1
STREAMER-MIB::sIndex.2 = INTEGER: 2
STREAMER-MIB::sIndex.3 = INTEGER: 3
STREAMER-MIB::sName.1 = STRING: nino
STREAMER-MIB::sName.2 = STRING: 01
STREAMER-MIB::sName.3 = STRING: informer
STREAMER-MIB::sClientCount.1 = Gauge32: 0
STREAMER-MIB::sClientCount.2 = Gauge32: 0
STREAMER-MIB::sClientCount.3 = Gauge32: 1
STREAMER-MIB::sRetryCount.1 = Gauge32: 0
STREAMER-MIB::sRetryCount.2 = Gauge32: 0
STREAMER-MIB::sRetryCount.3 = Gauge32: 0
STREAMER-MIB::sLifeTime.1 = Counter64: 6707983
STREAMER-MIB::sLifeTime.2 = Counter64: 6713981
STREAMER-MIB::sLifeTime.3 = Counter64: 1617905203800
STREAMER-MIB::sBitrate.1 = Counter64: 3571
STREAMER-MIB::sBitrate.2 = Counter64: 2580
STREAMER-MIB::sBitrate.3 = Counter64: 713
STREAMER-MIB::sBytesIn.1 = Counter64: 3043133741
STREAMER-MIB::sBytesIn.2 = Counter64: 2227175658
STREAMER-MIB::sBytesIn.3 = Counter64: 144049422
STREAMER-MIB::sBytesOut.1 = Counter64: 0
STREAMER-MIB::sBytesOut.2 = Counter64: 23894379
STREAMER-MIB::sBytesOut.3 = Counter64: 29263651
STREAMER-MIB::sStatus.1 = INTEGER: active(1)
```

```
STREAMER-MIB::sStatus.2 = INTEGER: active(1)
STREAMER-MIB::sStatus.3 = INTEGER: active(1)
STREAMER-MIB::totalClients.0 = Gauge32: 1
STREAMER-MIB::schedulerLoad.0 = Gauge32: 0
SNMPv2-SMI::snmpModules.1.1.6.1.0 = INTEGER: 1091569939
SNMPv2-SMI::snmpModules.10.2.1.1.0 = STRING: "Streamer"
SNMPv2-SMI::snmpModules.10.2.1.2.0 = INTEGER: 1
SNMPv2-SMI::snmpModules.10.2.1.3.0 = INTEGER: 6686
SNMPv2-SMI::snmpModules.10.2.1.4.0 = INTEGER: 484
SNMPv2-SMI::snmpModules.11.2.1.1.0 = Counter32: 0
SNMPv2-SMI::snmpModules.11.2.1.2.0 = Counter32: 0
SNMPv2-SMI::snmpModules.11.2.1.3.0 = Counter32: 0
SNMPv2-SMI::snmpModules.11.2.1.3.0 = No more variables left in this MIB View (It is past the end of the MIB tree)
```

This SNMP table contains variables related to *Flussonic* (`STREAMER-MIB`) and showing stream data, for example, stream name (`STREAMER-MIB::sName`), the number of stream clients (`STREAMER-MIB::sClientCount`), time while a stream is active (`STREAMER-MIB::sLifeTime`) and so on.

Streams are numbered .1, .2, and so on.

Here is the explanation of some variables that might not be obvious:

- `STREAMER-MIB::sStatus`

Returns integers that correspond to the following values:

```
* active = 1
* notInService = 2
* notReady = 3
```

- `STREAMER-MIB::schedulerLoad`

Consumption (in percentages, %) of the `Erlang scheduler` resource (average per last minute). Corresponds to the average value from **Pulse** > *Scheduler utilization for last minute*.

snmptranslate

To retrieve information about objects and identifiers (OIDs), use the `snmptranslate` utility with the `-Tz` flag:

```
snmptranslate -m /opt/flussonic/lib/mibs/STREAMER-MIB.mib -Tz
```

The utility produces a response similar to this one:

```
"org"                   "1.3"
"dod"                   "1.3.6"
"internet"              "1.3.6.1"
"directory"             "1.3.6.1.1"
"mgmt"                  "1.3.6.1.2"
"mib-2"                 "1.3.6.1.2.1"
"transmission"          "1.3.6.1.2.1.10"
"experimental"          "1.3.6.1.3"
"private"               "1.3.6.1.4"
"enterprises"           "1.3.6.1.4.1"
"streamerModule"        "1.3.6.1.4.1.36342"
"streamer"              "1.3.6.1.4.1.36342.1"
"streams"               "1.3.6.1.4.1.36342.1.1"
"streamsNum"            "1.3.6.1.4.1.36342.1.1.1"
"streamsTable"          "1.3.6.1.4.1.36342.1.1.2"
"streamsEntry"          "1.3.6.1.4.1.36342.1.1.2.1"
"sIndex"                "1.3.6.1.4.1.36342.1.1.2.1.1"
"sName"                 "1.3.6.1.4.1.36342.1.1.2.1.2"
"sClientCount"          "1.3.6.1.4.1.36342.1.1.2.1.3"
"sRetryCount"           "1.3.6.1.4.1.36342.1.1.2.1.4"
"sLifeTime"             "1.3.6.1.4.1.36342.1.1.2.1.5"
"sBitrate"              "1.3.6.1.4.1.36342.1.1.2.1.6"
"sBytesIn"              "1.3.6.1.4.1.36342.1.1.2.1.7"
"sBytesOut"             "1.3.6.1.4.1.36342.1.1.2.1.8"
"sStatus"               "1.3.6.1.4.1.36342.1.1.2.1.9"
"accounting"            "1.3.6.1.4.1.36342.1.2"
"totalClients"          "1.3.6.1.4.1.36342.1.2.1"
"serverStatus"          "1.3.6.1.4.1.36342.1.3"
"schedulerLoad"         "1.3.6.1.4.1.36342.1.3.1"
"streamerConformance"   "1.3.6.1.4.1.36342.2"
"streamGroup"           "1.3.6.1.4.1.36342.2.1"
"statGroup"             "1.3.6.1.4.1.36342.2.2"
"statusGroup"           "1.3.6.1.4.1.36342.2.3"
"security"              "1.3.6.1.5"
"snmpV2"                "1.3.6.1.6"
"snmpDomains"           "1.3.6.1.6.1"
"snmpProxys"            "1.3.6.1.6.2"
"snmpModules"           "1.3.6.1.6.3"
"zeroDotZero"           "0.0"
```

Objects description is provided in the `DESCRIPTION` fields of the `/opt/flussonic/lib/mibs/STREAMER-MIB.mib` file.

2.2.4 Devops

Flussonic in Kubernetes

This article provides some general information on *Flussonic* in *Kubernetes*:

- Essential terms and concepts
- Aspects of setting up Flussonic in Kubernetes

To test *Flussonic* in *Kubernetes* environment, use our media-server-operator guide

This document is relevant both for those who are new to *Kubernetes* and are willing to test *Flussonic* in this environment as well as for those who are already actively using *Kubernetes* in their business.

> **Note**
>
> This document does not cover the basics of *Kubernetes*, its configuration and usage. For this refer to Kubernetes Overview and Getting Started.

ESSENTIAL TERMS AND CONCEPTS

This glossary is intended for those who are unfamiliar with *Kubernetes* and are just beginning to learn about it. We will try to explain some *Kubernetes* terms and concepts to you so that you can look at them from a different perspective:

- ***Kubernetes*** (also known as *k8s*) is a cluster management program, a set of standards and rules that allows to manage a complex and dynamic cluster of microservices in a unified way. You could say that *Kubernetes* is a cluster operating system.
- ***Node*** is a worker machine (physical or virtual) in a *Kubernetes* cluster that executes containers.
- ***Pod*** is an instance of a program running in the *Kubernetes* OS. This program may consist of one or more containers. ***Pod*** is the smallest deployable unit of the *Kubernetes* ecosystem. A Pod is a group of one or more containers running on nodes in a *Kubernetes* cluster. This program may consist of one or more containers.
- ***Deployment*** is an object which declares the deployment type in *Kubernetes*. It makes sure that *Kubernetes* is running the required number of identical Pods. So, if you wanted to run multiple Pods, without Deployment you would have to manually define each Pod. Deployment is used to run stateless applications, that is, applications with no state tracking. Pods in Deployment are ephemeral, i.e. impermanent. This means that if a Pod is dropped and stops working, a new Pod will be started instead and it will not know anything about the Pod after which it was started. In Deployment Pods are interchangeable.
- ***DaemonSet*** is an object which declares the deployment type in *Kubernetes*. It ensures that each Node runs exactly one copy of a Pod. Its difference from Deployment is that DaemonSet ensures that each Pod is unique. In DaemonSet Pods have their own unique identifiers which are equal to the Pod hostnames. Thus, Pods are not interchangeable. DaemonSet implies that the same hostname is assigned to a Pod regardless of the number of Pod restarts. Knowing the uniqueness of a running Pod for a large number of Pods in a cluster is extremely useful. In the context of a *Flussonic* DaemonSet, this means that a *Flussonic* instance will be running with its own license key, on its own hostname, and with its own configuration.
- ***Volume*** is a catalog mounted in the Pod container.
- ***PersistentVolume*** is a disk space used to store data. PersistentVolume's (also referred to as PV) lifecycle is independent of that of a Pod using it. Therefore, if a cluster crashes, PV survives. In terms of *Flussonic*, PersistentVolume is great for recording and storing the archive. In the case of the cloud, cloud storage is provided as PersistentVolume. In the case of hosted servers, PersistentVolumes are disks on a particular node.
- ***PersistentVolumeClaim*** is a Pod request mechanism on PersistentVolume. PersistentVolumeClaims (also referred to as PVC) are specified in the same place as Pods. A Pod requests the storage through the PVC. Then PVC attempts to find suitable storage in a cluster.
- ***ConfigMap*** is a type of volume containing non-confidential data in key-value pairs. In terms of *Flussonic*, ConfigMap stores the static *Flussonic* configuration. ConfigMap can be defined as a file on disk or through environment variables.
- ***Secret*** is a type of volume (storage) containing confidential data, such as a username, a password, a license activation key, etc. In terms of *Flussonic*, Secret stores the `edit_auth` data: administrator login and password. Secret can be defined as a file on disk or through environment variables. Unlike ConfigMap, data in Secret is securely hidden.
- ***Service*** is an object that allows you to aggregate many Pods in one place at once. It gives access via a single entry point for different Pods. With Service you gain access to the Pod group.

ASPECTS OF SETTING UP FLUSSONIC IN KUBERNETES

Flussonic uses the data provided in the environment variables of the `env` field in the Pod configuration file `publish.yaml` to start. When it comes to sensitive data like admin login and password (`edit_auth` in *Flussonic*), *Kubernetes* recommends placing this information to the Secrets in Base64 strings. Credentials are pulled from the Secret to the environment (`env` field).

```
kind: Secret
metadata:
  name: test-secret
data:
# root:password
  edit_auth: cm9vdDpwYXNzd29yZA==
```

In order to start a Pod with your personal settings you should replace the default `root:password` values in the `edit_auth` variable with your Base64-formatted login and password.

The concepts of the configuration file for *Flussonic Media Server* and *Kubernetes* differ greatly. *Kubernetes* allows the creation of a *Flussonic* configuration file (`flussonic.conf`) from a directory of config files. Each of these config files represents a meaningful part of the *Flussonic* configuration file (`flussonic.conf`). Let's see how it is done.

```
kind: ConfigMap
metadata:
  name: streamer-presets
data:
  ports: |
    rtmp 1935;
  vod: |
    file vod {
      storage /opt/flussonic/priv;
    }
  publish: |
    template pub {
      prefix pub;
      url publish://;
    }
```

Here you can see the `ConfigMap` object (`kind: ConfigMap`) and with the *Flussonic* configuration in the `data` field. The configuration is broken down into multiple config parts (`ports` , `vod` , and `publish`). Replace the data in these config parts with your own if necessary. You can also specify the parts into which the configuration is divided.
Then the config parts are referenced in the `volumes` -> `configMap` field in the `items` section.

```
volumes:
      - name: config-templates
        configMap:
          name: streamer-presets
          items:
          - key: ports
            path: ports.conf
          - key: vod
            path: vod.conf
          - key: publish
            path: publish.conf
```

Each part is written in a separate `.conf` file and placed further in the directory of config files `/etc/flussonic/flussonic.conf.d` .

Flussonic k8s Media Server Operator

The media-server-operator is an extension for Kubernetes that simplifies the deployment of multiple instances of a media server with the same configuration. It is possible to run without the operator by manually configuring all necessary settings, after which you would need to monitor all changes and manually maintain new features of Flussonic.

media-server-operator addresses the following Kubernetes-specific issues:

- Organization of license data storage, allowing the media server to be restarted if it loses connection with the license server.
- Ensures that only one instance runs on a single server with the possibility of using direct network connections (HostPort).
- Configuration of all monitoring mechanisms and tracking of the server's startup.
- Convenient configuration of the configuration file, protected from being overwritten during operation.
- Creation of a service for balancing playback from a pool of media servers.

The operation of the media-server-operator is intended to be used in conjunction with the central-operator, which will ensure the delivery of stream configurations to each media server instance.

OPERATOR DEPLOYMENT

To utilize the operator, you simply need to apply a YAML file containing the CRD (Custom Resource Definition) to your system.

You require a running Kubernetes cluster and kubectl configured to interact with it.

```
kubectl apply -f https://flussonic.github.io/media-server-operator/latest/operator.yaml
```

For production installations, it's recommended to specify the version explicitly. A complete list of versions is available in the repository, for example:

```
kubectl apply -f https://flussonic.github.io/media-server-operator/24.3.3/operator.yaml
```

After specifying the version, you can proceed to launch media servers. Below is an example that will create a secret with the license and start the media server:

```
kubectl label nodes streamer flussonic.com/streamer=true
kubectl create secret generic flussonic-license \
    --from-literal=license_key="${LICENSE_KEY}" \
    --from-literal=edit_auth="root:password"
kubectl apply -f https://raw.githubusercontent.com/flussonic/media-server-operator/master/config/samples/media_v1alpha1_mediaserver.yaml
```

DISTRIBUTION

The operator is packaged for maximum convenience, suitable for use both in internet-connected clouds and in private networks. It consists of two parts:

- A YAML file with the k8s CRD description (these abbreviations stand for Custom Resource Definition, which is a specification for creating new types of objects in Kubernetes).
- A Docker image with our code on https://hub.docker.com/r/flussonic/media-server-controller.

These can be downloaded and stored locally for deployment.

External stream management

Flussonic provides tools for handling static and dynamic streams. It allows you to load static stream configurations and use live locations to publish streams with *dynamic names*.

> **Note**
>
> **Dynamic names** are *Flussonic* stream names unknown beforehand that *Flussonic* receives from the client. In a large system, stream names are unknown only to the *Flussonic* server but are known to some external subsystems. The external subsystem generates stream names, stores them, and returns them to the client on request. The client then reaches *Flussonic* with that name. Read more at Publish by dynamic name.

Thus, stream names can be:

- known to *Flussonic* in advance and specified in the configuration file (static streams),
- unknown to *Flussonic* in advance, but known to the external subsystem before accessing *Flussonic* (streams with dynamic names).

When the system consists of two or three servers, the mechanisms for static streams and streams with dynamic names work. If the system expands and the number of servers increases, these mechanisms start to break down, and issues arise.

ISSUES IN MANAGING A LARGE CLUSTER OF SERVERS

If the system includes 20 or more servers and works with many streams, both static and with dynamic names, the following issues arise:

1. It is unclear how to distribute the load between servers effectively.

   With a growing number of different streams, the system needs to run some streams with static configuration and others on client request. The mechanisms for static configuration work when the system is small and consists of one or two servers. When the system expands, the number of servers increases to 20, 30, or more, and the amount of content on these servers becomes even larger, it becomes unclear how to distribute streams between servers effectively. In this case, the usual mechanisms for static streams do not work anymore.

2. It may be necessary to log on to each server in a cluster to make configuration changes.

   A system that manages 20 or more *Flussonic* servers, capturing TV channels or IP cameras, should store the knowledge that a stream is captured and captured only once.

   The system should also store that a particular stream is captured by a particular server, like as server A. The system should also check from time to time that the stream is active and the signal is captured. If server A stops working, another server should capture the stream, like server B, and store that server B is now capturing the stream. If server B fails, another server has to capture the stream, and system should store the location of the stream. When a new server starts capturing the stream, the system should also check the statuses of the previous servers. If one of them starts working again, it is necessary to either:

   - remove that stream from the stream configuration of all the previous servers and leave it on the current server

   or:

   - return the stream to the server that it was on initially and remove it from the stream configurations of previous servers, avoiding stream capturing twice.

   Thus, if you need to make any changes related to stream configuration, you must bypass all servers in the cluster. If one of these servers is currently down, then when it comes up, it may be running with outdated settings. These settings can corrupt the source.

Flussonic solves the issues of capturing a stream twice, and failing of one of the capture servers with the cluster ingest mechanism (`cluster_ingest`). This mechanism allows you to automatically capture streams on another server when one of the servers in the cluster fails. It also removes streams from other servers when the initial one recovers. However, this mechanism solves problems in one way without the ability to customize it. So we came up with a solution to manage the configuration of streams using an external configuration backend called `config_external`.

ABOUT CONFIG_EXTERNAL

`config_external` is an internal *Flussonic* mechanism by which the server can download the current configuration of streams from the configuration backend. The **configuration backend or the configuration server** is an external resource that stores the stream management logic and is connected to the database to store that configuration.

HOW IT WORKS

The operation sequence of `config_external` differs depending on whether separate requests are implemented on the configuration backend to update static streams GET /update_streams_list and handle dynamic streams GET /dynamic_streams_list. By default, all requests are executed via a single method GET /streams.

Learn more about the two cases below.

One method for all streams (default)

The `config_external` works in cycles and updates the configuration every two to three seconds. Each configuration update cycle is as follows:

1. *Flussonic* requests the list of currently active static streams from the configuration backend via API (method GET /streams) and starts them if they have not started yet.
2. *Flussonic* calculates the difference between the list of stream names already running on the server and the list of static stream names returned by the configuration backend.
3. *Flussonic* sends an API request to the configuration backend with the resulting list of stream names (method GET /streams with `?name=..` in the query string) to request configuration for these streams.

> **Note**
>
> If the list of stream names is large enough, *Flussonic* will divide that list into parts of about a kilobyte each. Then *Flussonic* will request configuration for a part of the stream list until the end of list. It reduces the load on the configuration backend and the amount of traffic used.

1. The configuration backend returns the configuration for the requested list of streams. If the configuration backend has not returned the configuration for some of the requested streams, they are removed from the server automatically.

> **Warning**
>
> We **do not** recommend using one configuration server to serve all *Flussonic* servers in a cluster. The server won't be able to handle that number of requests and will be overloaded. To avoid this, we recommend making a full or partial data replication of the database on the local machine. If you use *Kubernetes*, this can be a sidecar container. This way, even if the connection between the central configuration server and *Flussonic* is lost, your system will be able to continue running, making your service more reliable.

Separate methods for updating static streams and querying dynamic streams

Flussonic requests the list of currently active static streams from the configuration backend via GET /streams method.

If the configuration backend returns `X-Config-Server-Separate-Endpoints: true` header in response, further requests to the configuration backend are sent as follows:

- The GET /streams method is called periodically, and if there are running streams on *Flussonic* server that are not included into the `GET /streams` response, then GET /update_streams_list is called to get configuration or delete such streams (similar to the default behavior).
- If a stream with an unknown (dynamic) name is requested from the *Flussonic* server, then GET /dynamic_streams_list is called to get the dynamic stream config.

  Several consecutive requests of this kind can be combined into one API call at *Flussonic*'s discretion with stream names listed in the `name` parameter.

> **Note**
>
> You can disable the use of dynamic streams by sending `X-Config-Server-Dynamic-Streams: false` header in response to GET /streams to reduce the load on the configuration backend. In this case, the *Flussonic* server will not accept requests for streams with an unknown name and will never route them to the configuration backend.

USAGE SCENARIOS

The configuration backend and `config_external` replace all mechanisms of managing the *Flussonic* cluster. They provide the ability to implement such mechanisms on your side.

With `config_external`, you can develop any logic of stream distribution between the servers in a cluster according to your needs. Powered by the configuration backend, you can implement your version of geo-targeted cluster ingest or the `source` mechanism for video retransmission. Here are the usage scenarios for you to consider:

Geo-targeted cluster ingest

The task is to capture the signal from the source in one country by one of the servers nearby.

The algorithm is as follows:

1. *Flussonic* requests the stream configuration from the configuration backend.
2. The configuration backend looks through the list of available servers and chooses the ones geographically closest to the source.
3. The configuration backend returns the configuration with a source ingest to one of those *Flussonic* servers.

Dynamic targeted republishing

The goal is to send a publish request to the least loaded transcoder.

You can implement the following operating logic:

1. The client sends a request to the publishing server.
2. The publishing server accesses the configuration backend and requests the stream configuration for the client.
3. The configuration backend looks through the list of available transcoders, identifies the least loaded of them, and returns a stream configuration with a `push` to the least loaded transcoder to the publishing server. There will be no loss of the first frame when the publishing server sends the stream to the transcoder.

CONFIGURING CONFIG_EXTERNAL

> **Warning**
>
> Do not manage streams with the Flussonic API and `config_external` simultaneously. For example, if you try to run the `Flussonic-API: PUT /streamer/api/v3/streams/{name}` with `config_external` enabled, *Flussonic* returns HTTP 400 error.

You can set the URL of the configuration backend in one of the following ways:

- add the `config_external` global option to the configuration file (`/etc/flussonic/flussonic.conf`):

```
config_external https://example.com/config_backend/streams;
```

- pass the `config_external` parameter in the API request (see the API Reference):

```
curl --request PUT --url http://127.0.0.1:80/streamer/api/v3/config \
--data '{"config_external":"https://example.com/config_backend/streams"}' \
--header 'Authorization: Basic base64_encoded_username:password' \
--header 'Content-Type: application/json'
```

- create an environment variable `STREAMER_CONFIG_EXTERNAL` and specify the path to the configuration backend as the value. We recommend using the environment variable only for the k8s environment (or other automatic deployment system); in other cases it is more convenient to use the first way of setting the option in the config file.

```
STREAMER_CONFIG_EXTERNAL=https://example.com/config_backend/streams
```

With this configuration, *Flussonic* will make two requests to the configuration backend every two or three seconds:

1. the list of currently active streams that should be running on a server,
2. settings for streams with dynamic names, if any.

> **Warning**
>
> If the configuration backend does **not** return the configuration for the requested stream, *Flussonic* will use the stream configuration from the configuration file stored on disk (`flussonic.conf`). Make sure that you do **not** use both configuration backend and configuration file stored on disk to configure streams. Otherwise, the server will **not** work correctly.

See also Config validation to learn how to find out if there are any errors in the configuration.

MANAGING EPISODES

`config_external` allows you to implement a mechanism for protecting certain parts of the archive from automatic deletion; you can use it to address the following objectives:

- To provide a nPVR (Network Personal Video Recorder) service, i.e. to save an archive with a recorded TV show broadcasted within a certain time period.
- To save an IP camera archive containing some important data (motion, face recognition, etc.)

We use *episodes* to protect the recordings. An ***episode*** is a set of metadata about the section of the archive including a unique identifier, the start time of the section, optionally the end time, etc.

The info about episodes is stored on a configuration backend server (*Flussonic Central* by default, see Protecting DVR sections from deletion). When *Flussonic* gets to a periodic task of the archive cleanup, it queries the configuration backend for a list of episodes for each stream and does not delete the parts of the archive that are covered by the episodes.

To enable the episodes:

1. Send `X-Config-Server-Episodes: true` header in response to GET /streams.
2. Set episodes_url in *Flussonic*'s DVR configuration.
3. Implement the response to the GET /episodes request on the side of the configuration backend to be returned at the URL you specify in `episodes_url`.

Config validation

Flussonic uses an OpenAPI scheme to validate its configuration file, utilizing the same rules as those applied to API calls. The configuration file is presented in a clear, human-readable format to facilitate manual editing by system administrators.

Flussonic checks the configuration file for any issues at the start-up and every time you change the config. If the format is invalid, for example an error occurs while saving or an option becomes unsupported after a software update, *Flussonic* will start anyway but will engage the crash mode to prevent disruptions to the system. In the crash mode, only **Config** and **Support** pages are available in the UI. You may also refer to the `streamer_status` (error code) and `text_alerts` (error description) parameters in the response to `Flussonic-API: GET /streamer/api/v3/config/stats` to check the config status.

It is not practical for *Flussonic* to validate the configuration file before starting the server, as the systemd service management system does not have a way to handle a service that is unable to function due to an invalid configuration file. This means that the service would simply restart without any clear indication about the issue, as systemd does not have a mechanism for signaling about this situation.

If the *Flussonic* configuration is generated by an external system rather than being manually edited, it is essential to validate the accuracy of the configuration file. The traditional method of generating the config, restarting the server, and checking for invalidity is not a practical solution for regular use.

Instead, we recommend using the JSON format for config generation by external tools to ensure proper functioning and avoid any issues.

To summarize this approach:

1. Create the config using the specified OpenAPI scheme (a format compatible with JSON schema).
2. Generate its JSON representation.
3. Validate it using any external validator according to the schema.
4. Write the resulting config to `/etc/flussonic/flussonic.conf`.
5. Start the server.

THE JSON CONFIG SCHEMA

You can get the current OpenAPI schema for Flussonic either:

- on our website
- or in the file `/opt/flussonic/lib/web-1/priv/schema-v3-public.json`, which is located on a server with the Flussonic package installed.

The schema defines the `#/components/schemas/server_config` type, which is used to validate the config when it is read. The text format is first translated into JSON and then validated using this schema. We recommend writing the config directly in the JSON format.

To examine the JSON representation of the config, you can use the utility provided with the Flussonic package:

```
# /opt/flussonic/bin/validate_config -j
{"listeners":{"http":[{"port":80}]}}
```

We recommend using the `jq` utility for working with JSON:

```
# /opt/flussonic/bin/validate_config -j | jq
{
  "listeners": {
    "http": [
      {
        "port": 80
      }
    ]
  }
}
```

The above example shows the JSON representation of the simplest possible config:

```
http 80;
```

JSON CONFIG VALIDATION

Config validation occurs in two stages:

1. Formal validation using the JSON Schema
2. Checking external conditions and those that cannot be expressed in the schema, such as the integrity of certificates

To validate the integrity of the config, you have two options: using an external utility for schema validation or using the `/opt/flussonic/bin/validate_config` utility provided with the Flussonic package:

```
# cat /etc/flussonic/flussonic.conf
htp 80;
# /opt/flussonic/bin/validate_config -j
{"col":1,"config":{},"detail":"htp","error":"unknown_command","line":1,"path":[]}
```

Similarly, with a JSON config:

```
# cat /etc/flussonic/flussonic.conf
{"listeners":{"http":[{"port":80,"flag":true}]}}
# /opt/flussonic/bin/validate_config -j
{"error":"unknown_key","path":["server_config","listeners","http",0,"flag"]}
```

2.3 Developers

2.3.1 TV

Authorization in Flussonic via middleware

MIDDLEWARE

A very important task that should be addressed when starting the OTT IPTV service is the limiting access to streaming servers. According to our statistics, many people never pay attention to it, and, consequently, overpay for the traffic: their streams are simply stolen.

Video may be distributed to everyone, but should be cleverly encrypted; keys should be distributed indiscriminately, it is called DRM. Another method of protection is limiting distribution of the video itself; this is called authorization.

In Flussonic, a very flexible authorization scheme is implemented that requires certain actions by Middleware.

The scheme of work is as follows:

- The client console requests the stream URL
- Middleware provides a URL with a unique token
- Flussonic uses this token to identify the session
- Upon opening a session, Flussonic checks this token with middlware

Such a three-link scheme is needed to avoid embedding authorization into Flussonic. In turn, Flussonic sends a request to middleware only once in a while, rather than at each request from the client.

The issue of choosing the proper token remains unsolved, and we can offer a couple of methods of generating it.

THE SHARE NOTHING TOKEN

The tokens may be generated to include all information that is necessary for authorization. For example, a token can be generated as follows:

```
token=sha1(secret_key + ip + stream_name)
```

After that, the token can be checked only if the secret_key is known. However, if an attacker tries to use this token, he will fail, since the IP will be different.

However, this token may be stored and used indefinitely. If a user has paid the subscription fee once, he may not pay again with this token.

Time may be inserted into the token:

```
time = utc()
token=sha1(secret_key + ip + stream_name + time)+":"+time
```

Now the middlware can check token age, and if it is more than one day old, it may be safely disabled. In practice, almost no one (except public TVs and fans of the Le Mans 24) is able to watch broadcasts for more than 24 hours in a row.

TOKENS IN THE DATABASE

Authorization may be combined with accounting for viewing, and a new unique token may be created each time the used starts viewing, populating it into the database:

```
token=uuid()
```

Later, in case of subsequent calls of flussonic to the middlware, the statistics for this session may be updated, storing the information about who watched videos and what volumes.

Export of EPG from MPEG-TS Streams

ABOUT GETTING EPG WITH FLUSSONIC

EPG (Electronic Program Guide) is an important part of any TV service. There are many ways to provide it to subscribers. For example, satellite TV transmits EPG data together with broadcasts in MPEG-TS streams.

Flussonic can extract EPG from the metadata of MPEG-TS streams received from a satellite receiver by UDP multicast. It imports EPG data to files that you can get via HTTP API. It then updates the EPG as it extracts new EPG data while receiving a stream, and you can get the updated EPG upon notification.

You can then export it to your IPTV middleware for providing it to subscribers. Also, the EPG in the JSON format is great for integration with web sites. This means that subscribers will receive EPG via the Internet as part of your paid services.

Flussonic exports EPG into two formats, each serving different goals:

- XMLTV. The standard format for describing TV broadcasts that is mostly used in IPTV middleware. It allows viewing the TV program and creating links to certain recorded broadcasts in the archive.
- JSON. These files have a structure specific to Flussonic. By using JSON files you can integrate with a web site and display the program on web pages.

Flussonic creates EPG for individual channels, for all channels, or a group of channels like Sport.

HOW TO GET EPG

Starting from version 20.03, you will need to explicitly enable EPG collection for a stream in the stream settings, with the option `epg on`:

```
stream channel5 {
  input tshttp://trancoder-5:9000/;
  input file://vod/epg.ts;
  epg on;
}
```

Alternatively, to turn on collection of the EPG via the UI:

1. Click the stream name in **Media**
2. Go to the **EPG** tab in the stream settings
3. Select the check box **EPG** and click **Save**.

With the EPG turned on, you can:

- Get EPG as an XMLTV or JSON file and then use these files in your services for subscribers.
- Subscribe to the event `epg_changed` to know when the EPG is updated and to receive updates.

  Updating the EPG means getting a newer file. Learn more in Events API about how to subscribe to events.

Important. Starting from Flussonic version 20.03, it is enough to access the stream at a special URL in order to get the EPG. The IPTV plugin is no longer used for this.

To get the EPG in the XMLTV format, use this URL:

- `/CHANNEL_NAME/epg.xml` — loads the EPG for a channel with the specified name.
- (deprecated) `/tv/channel/CHANNEL_NAME/epg.xml` — loads the EPG for a channel with the specified name (in versions prior to 20.03).
- (deprecated) `/tv/all/epg.xml` — loads the EPG for all channels (in versions prior to 20.03).

Format of the link for downloading an XMLTV file with EPG:

```
http://FLUSSONIC-IP/CHANNEL_NAME/epg.xml
```

To get the EPG in the JSON format, use the following URL:

- `/CHANNEL_NAME/epg.json` — loads the EPG for the channel with the specified name.
- (deprecated) `/tv/channel/CHANNEL_NAME/epg.json` — loads the EPG for the channel with the specified name (in versions prior to 20.03).
- (deprecated) `/tv/all/epg.json` — loads the EPG for all channels (in versions prior to 20.03).

Format of the link for downloading a JSON file with EPG:

```
http://FLUSSONIC-IP/CHANNEL_NAME/epg.json
```

IPTV

WHAT IS IPTV AND IPTV/OTT?

TV has become a huge part of our lives and now it's hard to imagine a home without it. Currently there are a few digital television services: satellite TV, cable TV, over-the-air TV and the recent ones — **IPTV** and **IPTV/OTT**. This article focuses on **IPTV**, how the TV signal is delivered to the viewer and how *Flussonic Media Server* can help in implementing such technologies.

IPTV AND ITS ARCHITECTURE

Internet Protocol television (**IPTV**) is the delivery of television content over *Internet Protocol* (IP) networks. This technology appeared in the late 90s to replace the traditional methods of TV signal transmission.

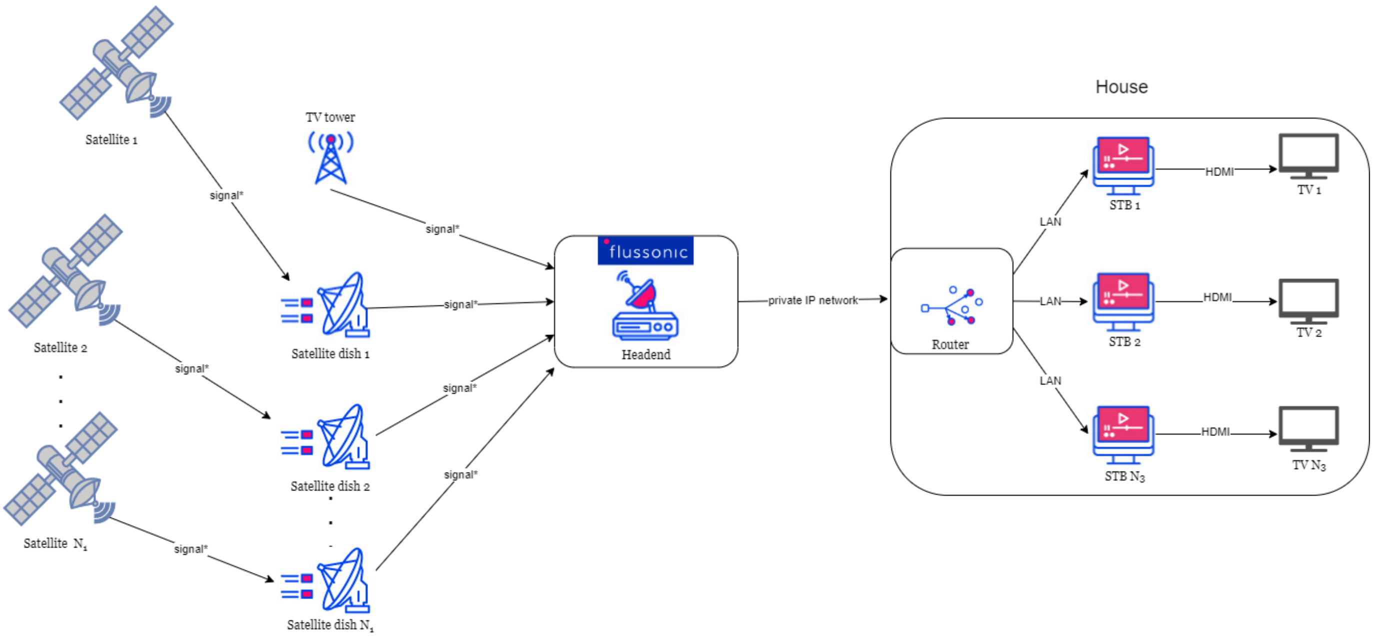
IPTV is a competitor to the conventional television content distribution like over-the-air broadcasting (*DVB-T/T2* in digital format), cable TV (*DVB-C/C2*) and satellite TV (*DVB-S/S2*) that are relatively simple to set up and affordable, but offer poorer variety of channel selection. Those types of broadcasting are inferior to **IPTV** in a number of features, which we will discuss later.

The classic example of **IPTV** service is that offered by an Internet provider. IPTV's great advantage in comparison with *DVB-T/T2* (short for *Digital Video Broadcasting — Terrestrial*) and *DVB-S/S2* (short for *Digital Video Broadcasting – Satellite*) is a wider selection of channels. If you have provider XYZ's dish installed on the roof, you get to watch only XYZ's TV channels. There aren't too many enthusiasts who would install 3 or 4 dishes from different providers, so a telephone company can offer a wider selection of channels in contrast with satellite TV.

It should be noted that traditional **IPTV** service uses *Internet Protocol*, a transport protocol to deliver the video content to the viewer through a cable. So that the operator/provider manages the stream delivery to the end-user. That does not correspond to delivery over the open-access network, i.e. Internet.

Traditionally, the term **IPTV** describes a specific list of technical solutions for receiving television signal and its retransmission to viewers. A classic **IPTV** architecture looks as follows (see *diagram 1.1*):

Diagram 1.1. IPTV architecture



Note 1: The **IPTV** scheme given above is a traditional one, so in every case it may undergo some changes.

Note 2: signal can be transmitted through various digital television broadcast standards: DVB, ATSC or ISDB.

Further in the article we will use the term *video content*. Let's agree that by this term we mean not only a *video stream*, but also an *audio stream* as well as *subtitles*, *closed captions*, etc., if any.

In the simplest case, the **IPTV** diagram includes a satellite dish, a *headend* and a set of *set-top boxes*.

Let's define some terms necessary for further understanding of the delivery of video content process:

Headend is a professional term for a satellite receiver that is capable of capturing a lot of TV channels from different sources simultaneously. A *headend* has three main functions:

1. Converting *DVB, ATSC or ISDB* signal into bytes.
2. *Descrambling*, i.e. decrypting it.
3. Sending this stream of bytes via *UDP* (*User Datagram Protocol*) multicast to the network.

Multicast is a method of data transmission to a group of recipients simultaneously. Note that *multicast* takes place only in the context of a private network or a *local access network* (*LAN*). *Multicasting* is similar to broadcasting, but it only transmits data to specific viewers and not to all of them. It is used to efficiently send streaming media and other content to multiple viewers at once by individual copies of the data.

For more information about sending multicast, see Sending multicast.

Set-top box (*STB*, "a box lying on top of the TV") is a small computer that contains a TV-tuner input and displays output to a TV set. A main device for controlling a *set-top-box* is a remote control.

Signal capture

Most **IPTV** operators use a satellite dish as a signal source to capture content due to its lower cost, but it is not the only possible source. In fact, there may be several sources of various kinds. For example, the *headend* can capture a signal from both satellite dishes and a TV tower at the same time (see *diagram 1.1*)

Capturing one TV channel using professional equipment should cost from roughly $100 to $1000 at a time. A dedicated Internet TV channel with a guaranteed quality costs about the same, but monthly. This is the reason why Internet TV is often provided without any quality guarantees. Sometimes a channel is captured via *SDI* (a cable transmitting raw original video). This is convenient, reliable and extremely expensive.

So, how is the signal transmitted via **IPTV**? The signal is transported according to a certain set of rules called *protocols* for devices to process the signal. Satellite transmits the *DVB-S/S2* signal to the satellite dish. Then content from satellite dishes (through same *DVB-S/S2* protocol) and/or local antennas (through *ISDB-T, ATSC or DVB-T/T2* protocols) is captured by the *headend* and converted to *IP* so that the router could transfer it to IP network. Stream is further transported to *STB* from the router, where it is tuned to be displayed on TV screen. *HDMI* cable is used to deliver the signal to TV.

A question may rise: why is **IPTV** better than a simple satellite dish (*DVB-S/S2*) if the operator installs the dish anyway?
Firstly, the operator installs not one plate, but 5 or 6, or even more, capturing all the channels that can only be reached, so that the subscriber gets a larger amount of various channels. Secondly, **IPTV** provides more different services. Thirdly, a significant part of the residents of apartment buildings in urban areas are not able to install a satellite dish, because of the fact that the signal from the satellite simply does not reach the dish. This can happen due to the following reasons:

- Typical for areas, where the distance between the buildings is extremely small. In this case, the signal's way from the satellite is blocked by the houses and the dish can not receive it.
- The windows of the apartment buildings face north. The satellites are placed in geostationary earth orbit above the equator. So, in the northern hemisphere they are visible only in the south. Hence, the signal simply cannot reach the dish.

Technically, it is possible to install a dish, but it just will not make any sense.

STB

Some *STB*'s can record and save live broadcasts for the viewers to watch later so they can playback and resume at their convenience.
It is important to acknowledge that recording of live TV broadcasts raises problems with the law. Many decades passed before the lawyers of content providers agreed to the use of the *videocasette recorder* (VCR) by the viewers. Thereby modern *set-top boxes* often just copy the meaningless and inconvenient functionality of old video recorders: recording a live broadcasting TV channel according to a preliminary schedule. In this case, a viewer has to preconfigure the *STB* to record at the right time.

First fairly primitive set-top boxes could only switch channels on a preloaded playlist. Modern consoles often come with web browsers like *Opera* or something based on *Webkit* (a free engine for displaying web pages), which are modified for video-specific tasks and processing the signal from the remote control. Usage of a web browser makes it easier to change the interface and add new features (for instance, buying content clicking a single button from the remote control). However, web browsers on slow *set-top box* processors are slower than some specialized applications, so there are still devices without web browsers on the market.

Middleware

To provide something more amusing and convenient than just a list of 300 channels that you need to scroll through from the first to the last, a new component comes in handy – *Middleware*.

Middleware is a separate component of the entire system, a software that provides additional services to users via *set-top boxes*. It should be noted that *Middleware* is not suited for some **IPTV** services and, hence, some *set-top boxes* receive a fixed list of channels.

With the help of *Middleware*, a viewer can quickly change the list of channels, classify channels by genre, access recorded live broadcasts, movies, enable the display of various information such as currency exchange rates, weather forecasts, etc.

That is how the first traditional **IPTV** model looks like. However, due to technological development this architecture has undergone some changes that leads us to the **IPTV/OTT**.

For more information about IPTV/OTT, see IPTV/OTT.

IPTV SOLUTION BASED ON FLUSSONIC MEDIA SERVER

So, we have examined what **IPTV** is, its way of content delivery to viewers. What part does *Flussonic Media Server* plays in this system and how can it be used to implement **IPTV**?

You can use *Flussonic Media Server* to create headend with its functionality: capturing the signal from the satellite dish and/or TV tower, descramble that signal and send it over IP network. *Flussonic* can also capture video streams from DVB boards directly. Furthermore, only one *Flussonic* server is needed to create a small 100-channel service.

Our product allows you to deliver the content the most efficient way possible and without loss of quality for viewer. So that you can focus on the content maker's and veiwer's experiences, while *Flussonic* will take care of the rest.

If you have any questions about implementing **IPTV** with *Flussonic Media Server* or you are willing to try out our product, please fill out the form to receive a free *Flussonic Media Server* trial key.

Our experts will contact you shortly, offer tech advice and consultation, and send you a trial license.

If you have not received an email from us within one hour, please check your "Spam" folder and add *Flussonic* to your "Trusted contacts" list.

Email: support@flussonic.com
Phone: +1 (778) 716-2080

IPTV/OTT guide

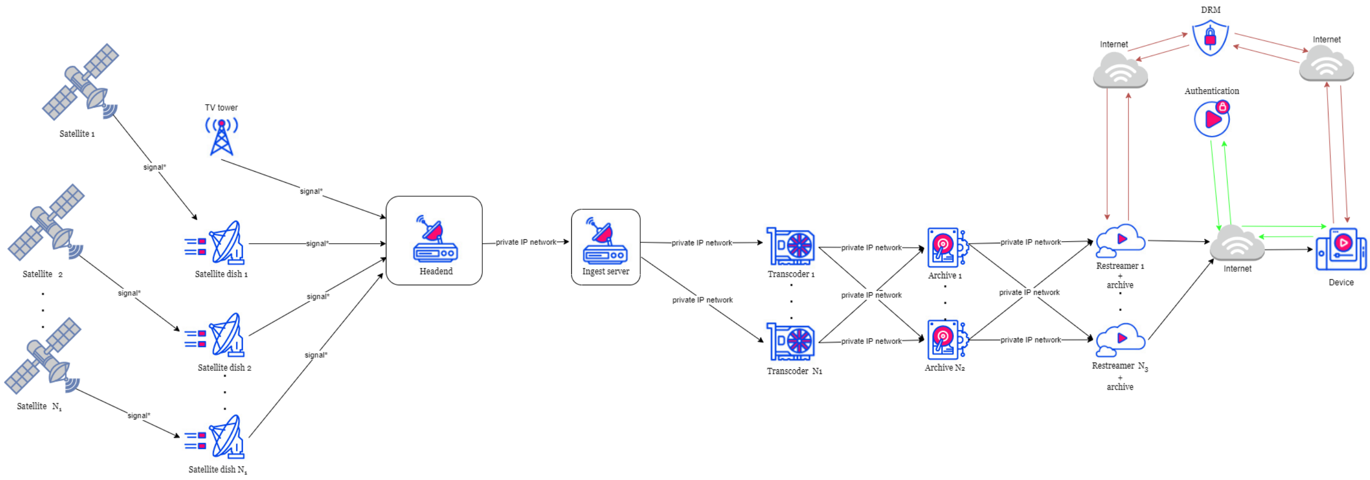
IPTV/OTT architecture

Over-the-top (OTT) is a means of providing television and film content over the internet at the request. It should be pointed out that it is not the internet provider, who provides the **IPTV/OTT** service and supervises it, unlike **IPTV**.
For example, capturing channels in Argentina, user himself can be in Germany and watch native channels, whereas his provider in Germany will not even know the list of provided channels.
This technology of TV signal transmission appeared about 10 years ago. At the moment, main providers and operators switch to **IPTV/OTT** due to its flexibility of convenience. However, the traditional **IPTV** model is still used, but mainly in the hotel and restaurant business.

One of **IPTV/OTT**'s main features is that it provides the content to the viewer directly via data network, in contrast with the traditional **IPTV** that provides the content through a private network managed by the provider.

Classic **IPTV/OTT** architecture looks like following (see *diagram 1.2*):

Diagram 1.2. IPTV/OTT architecture



This is how the signal transmission is performed in **IPTV/OTT**:

The first stage is the same as in **IPTV**: *headend* captures TV signal from a source or several different sources. Further stages of signal transmission will differ. Through the IP protocol it is then delivered to the *ingest* server. The output of the *ingest* becomes the input for the *transcoder* (see Transcoding), where the video stream breaks into 3 or more formats (depending on the quality of the input signal): *Full HD* (*1920×1080 pixels*), *HD* (*1280×720 pixels*), *SD* (*720x576 pixels*). The next step is to send this stream to *DVR*. *DVR* is a storage or an archive, where *video content* is recorded and stored. Right from the *DVR* the signal is transported to the *restreamer*, where the stream is encrypted to protect it from third-party users.

It should be kept in mind that *before the stream reaches the Internet, it is transmitted over the* private network*.

Before playing *video content* on any device (smartphone, PC, TV), passing authentication and getting an access to it is required. The *video content* is protected by the *Digital rights management* (*DRM*) system, so to get the access the viewer needs a decryption key (URL). After passing all the decryption and authentication stages, the viewer can enjoy the content.

IPTV/OTT model provides the following services:

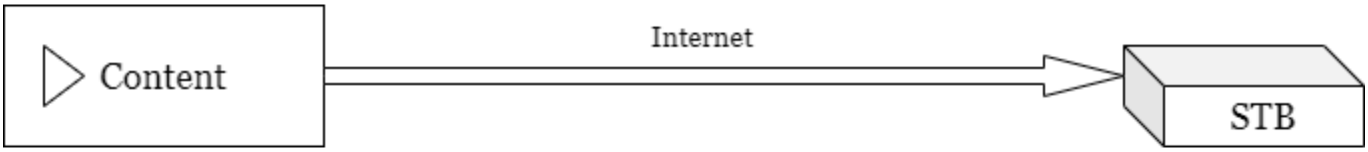
- *Video on Demand* (*VOD*). Individual delivery of video content to a subscriber or a viewer. It allows to watch any movie from the VoD server's media library.
- *Near Video on Demand* (*nVoD*). A pay-per-view video service intended for multiple users subscribed to *nVoD* service. The content broadcasting schedule is compiled beforehand and subscribers can look at the schedule and watch content of their interest.
- *Time-shifted TV*. Lets subscribers view live broadcasts later so they can playback and resume at their convenience. Rewind option is also provided for TV programs.
- *Transactional Video On Demand* (*TVoD*). Selected TV channels are recorded so they can be viewed whenever desired, but for a limited period of time (for example, a week).

Netflix, *Hulu* and *Disney +* are the examples of **IPTV/OTT**.

Transition from traditional IPTV to IPTV/OTT

It should be emphasized that **IPTV** and **IPTV/OTT** are two types of content delivery to the end-user. **IPTV/OTT** is considered to be a part of **IPTV** or, as to say, its new version. Roughly speaking this transmission path can be represented as follows (see *diagram 2.*):

Diagram 2. IPTV/OTT data delivery



For more information about IPTV, see IPTV.

In **IPTV** model the stream is transmitted via a *closed network*, while in **IPTV/OTT** model it is transmitted via an open-access network. Hence, the first difference is *access to the network*. In the first case (**IPTV**) — closed, in the second (**IPTV/OTT**) — open. The content in **IPTV** is almost impossible to intercept, so the level of piracy there is much lower than in the case of **IPTV/OTT**. Since this is an open network, it is much easier to intercept the content.

Next is *supervision of the signal transmission channel*. In **IPTV** the owner of the network is the same as the internet provider. This operator manages the entire process, i.e. knows how many users there are and what content they consume. Thus, there is a feedback. In **IPTV/OTT** there is no supervision and control over the signal transmission channel, it is not clear who is watching and what. So there is no feedback.
In **IPTV** the content consumer interacts *directly with the operator*, while in **IPTV/OTT** the consumer interacts *directly with the content producer*.

The next difference is *the quality of the transmitted material*. In the **IPTV** model the stream is passed on almost continuously and it is quite stable, which guarantees excellent quality, whereas the signal transferred in **IPTV/OTT** model is unstable and affects the quality of the content. Here we should mention *adaptive bitrate* or *ABR*. The aim of **IPTV** and **IPTV/OTT** is to deliver the content without visible failures and delays for the viewer. Thus, given the fact that in the **IPTV/OTT** model the signal may be unstable (due to the speed and the quality of the internet connection), **IPTV/OTT** technology adjusts to the current network performance, so that the video and audio are delivered without pauses.

IPTV is characterized by georeferencing. The delivered content is specific for the place where it is distributed. **IPTV/OTT** provides all kinds of content to the viewer despite his location.

Considering the *price*, it is necessary to bear in mind the following: the *cost of services* and *how it is formed*. Let's start with the cost: **IPTV** is more expensive than **IPTV/OTT**. The cost of **IPTV** is usually formed by the cost of the following package: internet access + the service itself **IPTV** (i.e. connecting the *STB* and its maintenance). The cost of **IPTV/OTT** is equal to the cost of the internet access service. **IPTV/OTT** is cheaper than **IPTV** because it uses free content from public channels.

It should also be noted that *new content release* is quicker in **IPTV/OTT** than in **IPTV**.

To sum up, all the main points are tabulated (see *table 1*):

Table 1. IPTV and IPTV/OTT

Features	IPTV	IPTV/OTT
Transmitted content quality	**+** (high)	**+/-** (depends on the network performance)
Transmission channel supervision	**+**	**-**
New content release	**-**	**+**
Price	**-**	**+**
Price components	internet (IPTV included)/internet + IPTV	internet + subscription
Connection reliability	**+**	**-**
Network access	**-** (closed)	**+** (open)

In conclusion, classic architectures of **IPTV** and **IPTV/OTT** were reviewed. Nowadays **IPTV** and **IPTV/OTT** technologies have become less distinguishable and there is no clear line between the two. Although the main difference remains the same: *the way of delivering the stream to the end-user* (the last stage). **IPTV** model uses private network and **IPTV/OTT** — open network, i.e. internet.

KEY FEATURES OF THE IPTV SERVICE IMPLEMENTATION

Providers of the **IPTV/OTT** services face challenges that were not present 5 or 10 years ago. Let's have a look at them and define the aspects of the **IPTV/OTT** service implementation.

Capturing and transcoding

Satellite equipment is notoriously resistant to technology updates. Historically, satellite TV uses the *MPEG-2* video and, so to speak, *MPEG-2* audio codecs. The implementation of the *H.264* codec to satellite broadcasting has been going for years and has not finished yet.

However, neither of those are supported by modern devices like iPhone and others. Moreover, the *H.264* signal sent from satellite today cannot be processed by the iPhone due to the intra-refresh technology.

The *MPEG-2* codec can be safely replaced with *H.264* to achieve 3-4 times more bitrate efficiency and, consequently, traffic economy.

When *HD* signal is captured from the satellite and delivered to viewers over a non-local network, bandwidth limitations can prevent most users from consuming the content, so the signal should be encoded in different bitrates to enable adaptive quality switching.

Accordingly, video and audio coming from satellite needs to be transcoded to *H.264/AAC*, since iPhone doesn't support it. *HD* signal needs multi-bitrate conversion.

How the issues of capturing and transcoding are solved by *Flussonic Media Server*?
Flussonic Media Server can receive video over IP protocols not only from any IRD (Integrated Receiver-Decoder) devices or systems, but also directly from DVB-S and some other cards.
Flussonic Media Server can also decode video from *UDP/HTTP, MPEG-TS, RTMP* sources and encode it in multiple formats, that allows to play videos not only on *set-top boxes*, but on tablets and iPhones as well.

From catch up (programs archive) to Interactive TV

As previously mentioned, historically, *set-top boxes* have a feature of recording just one TV channel's broadcast on demand. This approach doesn't work well, since people often forget to set the recording timer and then get frustrated: what is the reason to buy this expensive *STB* if it's no better than any old *VCR*?

The modern approach to providing access to the archive of TV shows is as follows: record the entire TV broadcast on the provider's side and give the viewer permission to manage the watching itself, namely: * watching the programs from the archive using the TV schedule or *EPG* (Electronic Program Guide), * rewind, * pause.

To provide the *Interactive TV* service, following steps should be : * implement archive on the provider's side * configure the players on the viewer's side.

Flussonic Media Server provides a wide functionality range to work with the archive, using *DVR* (Digital Video Recording) technology. Such as: user-friendly navigation and access to the archive, unlimited recording space, quick preview of individual thumbnails without a need for rewinding and etc.

For more information, see DVR

Linear TV broadcasting over Wi-Fi

The conventional way of multicast delivery has to deal with interference caused by *Wi-Fi*. *HD* signal (*6-15 Mbit/s, compared to the old SD's 1-3 Mbit/s*) and home Wi-Fi become a challenge for multicast: an expensive TV set shows the tell-tale green squares (pixels) instead of a crystal clear picture. It happens due to considerable packet loss on the way from the *headend* system to the *set-top box*.

Flussonic Media Server can function as a restreamer and perform multi-stream broadcasting, allowing to configure several signal sources and set up a fail- safe configuration.

For more information, see Cluster restreaming

Geo-distributed delivery

As the number of the **IPTV** service's subscribers increases, sooner or later the provider faces a challenging situation when delivering content from one central server becomes tricky or almost impossible.

Typical examples are: - provider opening a branch office in another city - a massive influx of new subscribers in another country as a result of an ad campaign.

In situations like these delivering video content from one central server becomes impractical, especially if there appear to be clusters of viewers located close to one another watching the same TV channel.

In order to save traffic, local retranslator servers are used: the channel's content is transmitted from the central repository to the local retranslator and then sent to the end-users located nearby.

This architecture may become far more complex with the increasing number of retranslators and channels. Since every channel must be set up manually, the administrator has to deal with a vast number of channels manually.

Also, geo-distributed video delivery sets its own limitations to archiving. It is not feasible to store the past content of all channels on each local server. In fact, the content of channels with narrow audience should be stored on one central server. And yet, every subscriber must be able to access this archive.

Taking geo-distributed video delivery into account, the question of access to the archive arises: does it make sense to store all recorded TV broadcasting channels on all servers? Of course not. It is easier to store rarely watched channels in central archive, but the access to this archive has to be provided for the viewer.

Flussonic Media Server offers a number of tools to solve those problems.

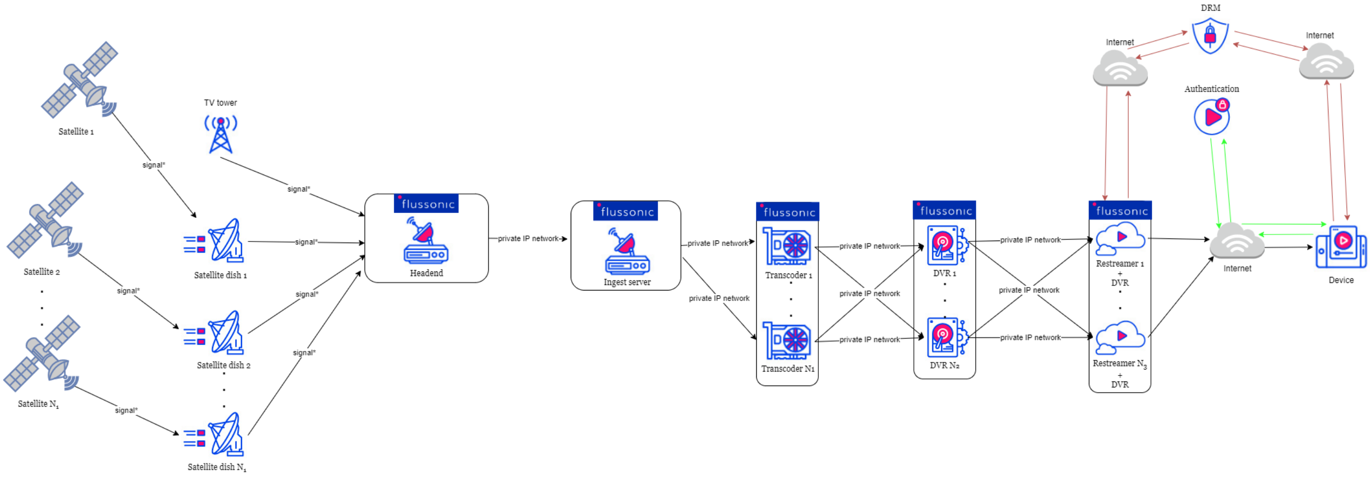
For more information, see DVR and Cluster restreaming

IPTV/OTT SOLUTION BASED ON FLUSSONIC MEDIA SERVER

So, we have examined what **IPTV/OTT** is, its way of content delivery to viewers as well as the transition from **IPTV** to **IPTV/OTT**. Furthermore, we have also covered key features in this area. What part does *Flussonic Media Server* plays in this system and how can it be used to implement such technology?

Flussonic Media Server may be used in different stages of content delivery from capturing the signal from the satellite dish and/or TV tower by the headend to its playback by the end-user. Thus, this entire segment of the path can be implemented with *Flussonic Media Server* (see *diagram 3.1*).

Diagram 3.1. Flussonic Media Server in IPTV/OTT



In the case of **IPTV/OTT**, each individual component (*headend, capture server, transcoder, DVR, and restrimer with DVR* function) can be implemented with *Flussonic*. Our product allows you to deliver the content the most efficient way possible and with minimal loss of quality for viewer. So that you can focus on the content maker's and veiwer's experiences, while *Flussonic* will take care of the rest.

If you have any questions about implementing **IPTV/OTT** with *Flussonic Media Server* or you are willing to try out out product, please fill out the form to receive a free *Flussonic Media Server* trial key.

Our experts will contact you shortly, offer tech advice and consultation, and send you a trial license.

If you have not received an email from us within one hour, please check your "Spam" folder and add *Flussonic* to your "Trusted contacts" list.

Email: support@flussonic.com

Phone: +1 (778) 716-2080

2.3.2 Internet streaming

Setting up CDN

This article will teach you how to streamline the delivery of a video content to other continents. The video content is both live video restreamed from the ingest servers and video stored on HDD (e.g. DVR or VOD) at origin servers.

The main problems with the delivery of such a content over long distances via the public Internet are:

- **Instability**. The signal goes through a lot of routers to get from an ingest/origin server to the end user while the stability of data transmission is not guaranteed on that path. It may take dozens of seconds before the user receives the requested video.
- **Costs**. The channels between continents, countries, or regions are usually more expensive than local/city channels so referring distant users directly to ingest/origin servers is quite unprofitable.
- **Load**. A simultaneous video request by a large number of users can overload your server.

This problems are usually solved with a **CDN** (Content Delivery Network). It is a set of servers with specialized software that are physically located in regions where the users access your content. You may fully duplicate or cache the DVR content on the local CDN servers and distribute the live playback sessions between them. This mechanism significantly speeds up the content delivery and distribution to the end user, reduces the cost of communication channels and optimizes the load on the servers.

Flussonic has a number of features to simply implement a CDN of any scale. The use of *Flussonic* as the specialized software on CDN servers provides a number of advantages, for example:

- The built-in caching mechanism designed for video delivery with personalized advertising (see also Content monetization using ad insertion).
- You don't have to send several protocols to the edge servers via expensive channels because *Flussonic* on the edge server will pack the video into the required format.

In this article, we will consider an example of a small CDN broadcasting live shows.

Below you will find configurations of all servers in the video pipeline, from ingest to edge (CDN). If you are only interested in CDN, please refer directly to Distribution.

THE PIPELINE

The pipeline structure will be as follows:

- In the capture region, there will be at least two redundant servers.
- In the region of broadcasting, the servers will ingress video from one of the two sources.
- Each channel will be transmitted between the regions only once, to keep the intercontinental traffic to minimum.
- In the capture region, video will be transcoded and recorded in order to prevent losses in case of channel outage.

In the examples below, each server will perform a certain role like ingest and transcoding, recording, distribution. You can combine the servers in the capture region in any way you want depending on the hardware resources available, for example, ingest, transcode and record on the same server. The diagram below shows the video streams in the proposed system, dotted arrows indicate backup channels.

digraph { rankdir="LR";

tvchannel1 -> transcoder1; tvchannel1 -> transcoder2 [style=dashed]; tvchannel2 -> transcoder1; tvchannel2 -> transcoder2 [style=dashed];

transcoder1 -> dvr1; transcoder1 -> dvr2; transcoder2 -> dvr1 [style=dashed]; transcoder2 -> dvr2 [style=dashed];

dvr1 -> edge1 [label="Restreaming, M4F x No. of channels"]; dvr1 -> edge2 [label="Restreaming, M4F x No. of channels"]; dvr2 -> edge1 [label="Restreaming, M4F x No. of channels", style=dashed]; dvr2 -> edge2 [label="Restreaming, M4F x No. of channels", style=dashed];

edge1 -> user1 [label="Play, DASH x No. of users"] edge1 -> user2 [label="Play, HLS x No. of users"]

subgraph cluster_capture_region { label = "Capture region";

```
tvchannel1 [label="TV channel 1"];
tvchannel2 [label="TV channel 2"];
transcoder1 [label="Transcoder 1", shape=box];
transcoder2 [label="Transcoder 2", shape=box];
```

```
dvr1 [label="DVR 1", shape=box];
dvr2 [label="DVR 2", shape=box];
```

}

subgraph cluster_cdn{ label = "Edge servers"

```
edge1 [label="CDN server 1", shape=box]
edge2 [label="CDN server 2", shape=box]

user1 [label="User 1"];
user2 [label="User 2"];
```

}

}

Using this scheme, we will show *Flussonic*'s capabilities.

See also the detailed article about video pipeline in our blog.

INGESTING AND TRANSCODING

Various configurations may be made for ingesting streams depending on whether the video may be taken from the source several times, or not. In the easiest case, if you have a video coming from a headend in a multicast via UDP, you can just configure the capturing of the same video on several servers (see Multicast Receiving). However, we will consider the example with cluster ingest which is good for capturing from a source with an expensive/slow connection.

To make sure clients with different bandwidth receive the video sustainably, you have to transcode the video to a multibitrate stream. Thus, the users will be able to select bitrate (i.e. quality) that will be suitable for them. You can transcode on the ingest servers or dedicate a separate pool of servers for that. It depends on the available resources because transcoding is a resource consuming process. In our example, we will perform transcoding on the ingest servers.

We will configure a pool of two transcoders (`transcoder1.example.com` and `transcoder2.example.com`) that backup each other using the cluster ingest mechanism:

Transcoder 1

```
cluster_key mysecretkey;

# Remote sources:
peer transcoder1.example.com {}
peer transcoder2.example.com {}

# Ingest streams:
stream tvchannel1 {
  input udp://239.0.1.1:5500;
  transcoder vb=2048k preset=veryfast ab=128k vb=1024k preset=veryfast ab=64k;
  cluster_ingest;
}
stream tvchannel2 {
  input udp://239.0.1.2:5500;
  transcoder vb=2048k preset=veryfast ab=128k vb=1024k preset=veryfast ab=64k;
  cluster_ingest;
}
```

Transcoder 2

```
cluster_key mysecretkey;

# Remote sources:
peer transcoder1.example.com {}
peer transcoder2.example.com {}

# Ingest streams:
stream tvchannel1 {
  input udp://239.0.1.1:5500;
  transcoder vb=2048k preset=veryfast ab=128k;
  cluster_ingest;
}
stream tvchannel2 {
  input udp://239.0.1.2:5500;
  transcoder vb=2048k preset=veryfast ab=128k;
  cluster_ingest;
}
```

Here and further on, we agree that the servers have correct hostnames that can be resolved.

All the servers should have the same cluster key. In our example it is `mysecretkey`, but it could have any value.

Note that the configuration is completely identical on both servers. This is a requirement of cluster ingest mechanism: all streams must be declared on all servers.

DVR RECORDING

The servers on which the archive is stored are called *origin* servers (`origin1.example.com` and `origin2.example.com`). In our example those servers restream the video from the transcoders to the CDN edge servers while recording an identical archive so that a backup copy could be accessed if one of the servers fails.

DVR 1

```
cluster_key mysecretkey;

# Remote sources:
source transcoder1.example.com {
  dvr dvr/origin1 2d;
}
source transcoder2.example.com {
  dvr dvr/origin1 2d;
}
```

DVR 2

```
cluster_key mysecretkey;

# Remote sources:
source origin1.example.com {
}
source transcoder1.example.com {
  dvr dvr/origin2 2d;
}
source transcoder2.example.com {
  dvr dvr/origin2 2d;
}
```

With this configuration, *Flussonic* will pick up all the channels from one or another transcoder and write them locally to the archive.

DISTRIBUTION

The CDN servers will act as restreamers retranslating all the streams from the previous link in the pipeline (DVR servers in our case) and caching the recent requests to the DVR:

CDN server 1

```
cluster_key mysecretkey;

source origin1.example.com {
  cache /storage/cache 2d;
}

source origin2.example.com {
  cache /storage/cache 2d;
}
```

CDN server 1

```
cluster_key mysecretkey;

source origin1.example.com {
  cache /storage/cache 2d;
}

source origin2.example.com {
  cache /storage/cache 2d;
}
```

The restreaming mechanism implemented by Flussonic Media Server ensures that each stream is transmitted to each edge server only once.

The requests to the archive are transparently routed to the corresponding DVR server while the edge servers act as proxy servers. The received videos are cached on the edge servers. Thanks to the cache, subsequent requests for the same content are served faster. Read more about cache here. See also Cluster DVR to learn more about accessing the archive in a distributed video delivery environment.

Since the CDN is the last node in the pipeline before delivering streams to clients, you can add the authorization here and/or define the list of allowed countries.

Scaling the CDN

In case of distributing a large amount of video content, you will probably have several CDN servers in each distribution region so there will be a need for load balancing. In that case you should configure a Flussonic's load balancer in each region like that:

digraph { node [shape=box] rankdir="LR"; dvr [label="DVR"]

dvr -> edge1 [label="M4F restreaming"] edge1 -> balancer1 [label="load data", style=dashed] edge2 -> balancer1 [label="load data", style=dashed] client1 -> balancer1 [label="1. play request"] balancer1 -> client1 [label="2. HTTP redirect to edge"] client1 -> edge1 [label="3. play request"] edge1 -> client1 [label="4. live/DVR playback"] edge1 -> dvr [label="3.1 DVR?"] dvr -> edge1 [label="3.2 M4F DVR"]

subgraph cluster_region1 { label="Region CDN" edge1 [label="Edge 1"] edge2 [label="Edge 2"] balancer1 [label="Balancer"] client1 [label="Region client"] }

}

The diagram shows how live video is being restreamed from the DVR server, and the archive is also transmitted when the user accesses an uncached DVR. The clients receive all the video from the edge servers and do not know about the existence of DVR and other pipeline links; the balancer only performs redirects, i.e. video is not streamed through it.

Flussonic UGC implementation guideline

This document explains how to design and implement a UGC service or a platform with *Flussonic Media Server*. *Flussonic Media Server* is a multi-purpose software solution for launching a high-load video streaming service of any scale.
This document is intended for:

- Software architects
- CTOs
- Product or project managers of the companies

of the companies that want to develop their own UGC streaming platform.
This document doesn't include information about installing or configuring the *Flussonic Media Server*. For information on installing *Flussonic Media Server* and getting started with it, see Installation and Quick Start.

Introduction focuses on the objectives that *Flussonic Media Server* can solve to create a UGC streaming platform and where UGC streaming platforms are used. The architecture of a typical UGC streaming platform describes the architecture design of a UGC platform or a service, its key components, workflow, and ways to make your system resilient and scalable. Analyze and establish requirements examines requirements for the service. Design the network architecture shows how to estimate the network bandwidth for the service and what you need to create an architecture design diagram. Example of an implementation strategy for a UGC service describes a real-life project.

TABLE OF CONTENTS

INTRODUCTION

UGC is both:

- Streams with ultra-low latency within one second, such as group conversations in Google Meets or Zoom
- Streams in the social networks with a latency of several seconds

Streaming UGC platform is used for streaming the content like video games, business events or webinars.

UGC streaming platforms are used in the following fields:

- **Gaming**. Live streaming of tournaments and competitions in cybersports, walkthroughs and reviews of video games.
- **Sports**. Live streaming and recording of tournaments and competitions in various kinds of sports.
- **Education**. Live streaming and recording of lectures, seminars, webinars, courses and specializations.
- **Religion**. Live streaming and recording of worship services.
- **Events**. Presentations of new mobile phone models, streaming of cultural events.

The project starts with defining the goals and objectives.

Project **goal** is the result we want to achieve that adds value to the business; the purpose of the project. For example, to increase sales revenue or reduce company expenses.
Project **objectives** are specific steps or tasks to pursue the goal.

Flussonic Media Server allows you to solve the following objectives when creating a UGC service or a platform:

- Receiving streams from creators with the highest possible quality and stability.
- Ensuring the hardware is used under uneven load during the time of day.
- Ensuring even distribution of network traffic among servers.
- Charging creators for retransmitting and storing the recordings.
- Recording, storing, and playing back the recordings from the archive to viewers.
- Multistreaming content to social networks.
- Providing a comfortable environment for online video and audio conferences without the need to install any additional applications.
- Charging viewers and paying royalties to creators.

Without establishing goals and objectives, you can not properly specify the requirements for the project and build a diagram of the service architecture.

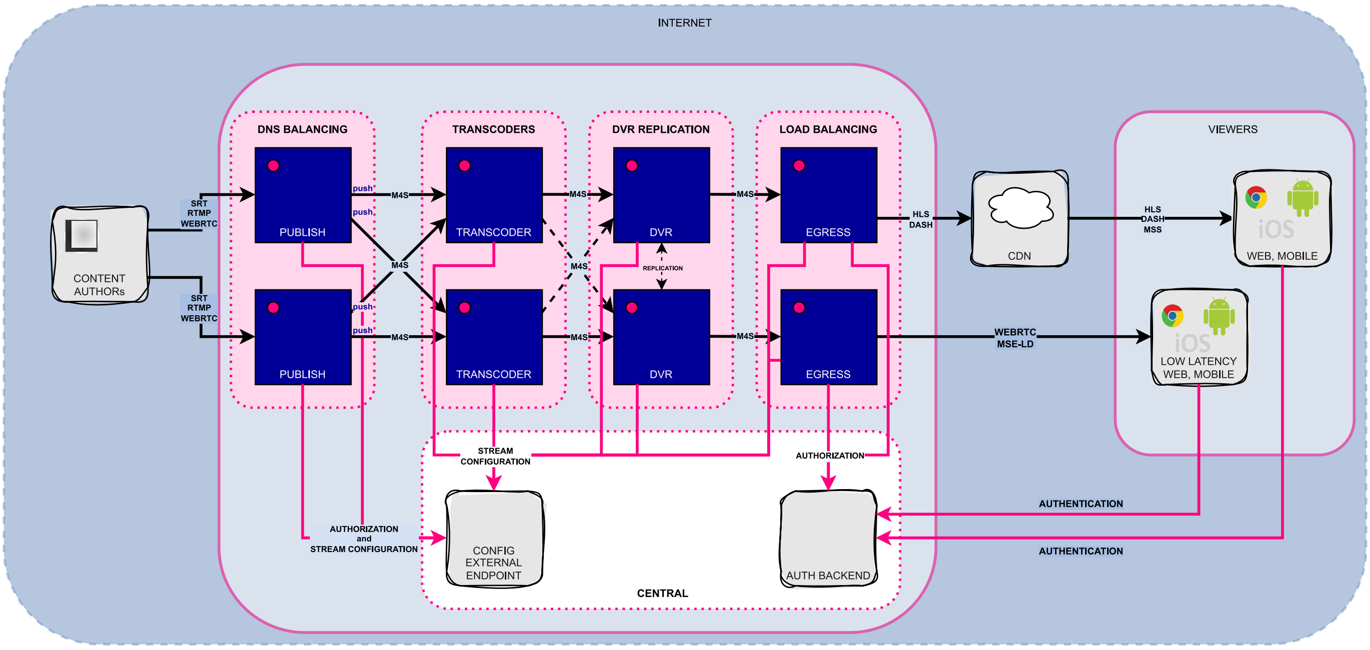
THE ARCHITECTURE OF A TYPICAL UGC STREAMING PLATFORM

This chapter covers the following topics:

- What components make a UGC streaming platform and what tasks they solve
- How a stream is delivered to viewers
- How to make your system resilient and scalable

The following diagram shows an architecture of a typical UGC streaming platform built with *Flussonic Media Server*:

Diagram 1. UGC platform architecture design



Here:

- Publish—publishing server
- Transcoder—transcoding server
- DVR—DVR server
- Egress—egress server
- Central—*Flussonic* management server
- Content authors—devices of content creators
- Viewers—devices of viewers

Key components

Here are the key components that make up a UGC streaming platform:

- Publishing server is a server that receives a stream from the creator.
- Transcoding server is a server that transcodes a stream into an array of streams at different resolutions and bitrates. Optional component.
- DVR server is a server that takes incoming video streams, records, and stores them, then sends them on demand. Optional component.
- Egress server is a server that delivers the content to the viewers.
- Central ia a management server in *Flussonic* that provides a central point of access for managing server configurations for every server.
  - Authorization backend is a tool for user authorization.
  - Config_external endpoint is a mechanism for managing stream configuration.

Publishing server

Publishing server allows to solve the following objectives:

- Receiving streams from creators with the highest possible quality and stability.
- Multistreaming content to social networks.

Flussonic receives live streams over various protocols:

- WebRTC
  - Requires no additional software installation to start streaming. A creators only needs a browser and an Internet connection.
  - Allows to automatically adjust the video quality to the speed of the Internet connection using the adaptive bitrate (ABR) algorithm. The higher the bitrate, the better the audio and video quality for the creator.
- SRT
  - Provides low-latency and uninterrupted publishing by reducing video quality.
- RTMP
  - Allows creators to use almost any video editing equipment and open source solutions.

Publishing server repackages the stream to the Flussonic-to-Flussonic streaming protocol M4S* and then transports it to one of the servers:

- the transcoding server
- the DVR server
- the egress server

depending on the architecture design and the objectives.

> **Note**
>
> M4S is a real-time streaming protocol for transmitting the video only between *Flussonic* servers. The protocol supports all *Flussonic* codecs. To learn more about M4S, refer to M4F and M4S protocols.

Read also:

- Publishing a stream from OBS Studio to Flussonic Media Server

Transcoding server

The transcoding server prepares the content to deliver it at the best possible quality to viewers:

- To create multi-bitrate stream.
- To balance the load between the transcoding servers.
- To monitor the volume of the transcoding traffic.

The transcoding server receives the stream and transcodes it, getting a multi-bitrate stream at the output.

Creating multi-bitrate stream

To prepare the content, prepare the multi-bitrate streams—synchronized video tracks with different resolutions and bitrates. For the transcoder to prepare the stream for all viewers, the input stream should be at the highest bitrate settings.

Here are the optimal video profiles for the video stream delivery:

- 1920x1080p
- 1280x720p
- 896x504p

Also client device determines the profile for video delivery. For example, for the majority of smartphones 896x504p profile is enough, however, TVs require at least 1920x1080p. The transcoder can both downscale and upscale the resolution of the stream to deliver it to viewers. For example, a creator publishes a stream at 720p resolution, and the transcoder upscales the resolution to 1080p to deliver it to viewers.

> **Note**
>
> The higher the stream resolution and bitrate, the greater the latency.

If you receive the streams from various sources simultaneously, such as a browser, a webcam, OBS, or a video encoder, transcode each such WebRTC, RTMP, and SRT stream, rather than repackage it. WebRTC, RTMP, and SRT have different and not fully compatible codecs.

The transcoder also applies the following delays:

- Three to ten seconds to achieve the highest quality at minimum bitrate.
- One to two seconds to achieve acceptable quality with widely varying bitrates.

Load balancing of the transcoding servers

To distribute the load between the transcoding servers and prevent the service from stopping, use the external stream management with the config_external mechanism. This way you don't need to configure the streams manually.

Monitoring of the volume of the transcoding traffic

The volume of the transcoding traffic is the sum of bitrates of tracks in the stream profile. With this value, you can determine the cost of transcoding for each creator.

Based on the source stream bitrate, it can be a combination of profiles. For example, if the 1920x1080p source stream is at 5 Mbps bitrate, then recommended profiles are 1920x1080p at 4 Mbps, 1280x720p at 2 Mbps, and 896x504p at 900 Kbps. Then the volumes of the transcoding traffic are:

- For 1920x1080p: 4 000 Kbps + 192 Kbps = 4192 Kbps
- For 1280x720p: 2 000 Kbps + 128 Kbps = 2128 Kbps
- For 896x504p: 900 Kbps + 96 Kbps = 996 Kbps

Transcoding can be done with a specialized dedicated hardware, a CPU or an external or integrated GPU. *Flussonic Media Server* has a built-in transcoder. It supports transcoding with a GPU or a CPU. To transcode streams, you can use media server or Flussonic Coder.

Transcoding is a computationally expensive process, and it puts a heavy workload on the CPU and GPU. Changing video codec, resolution, or bitrate requires a high amount of system resources, unlike *transmuxing*. Transcoder isn't required for transmuxing.

Transcoding and transmuxing operate on different levels: transcoding—content or data, transmuxing—container and streaming protocol.

> **Note**
>
> If you need an expert advice and assistance when to choose the right equipment for your project, contact our technical support team at support@flussonic.com.

For more information about transcoding and supported codecs and protocols, see:

- Transcoding
- Supported protocols and codecs

DVR server

DVR server records live streams and stores the recordings of them in an archive.

DVR server provides your viewers an opportunity to pause, rewind, fast-forward, re-watch, and record any live stream and watch it later.

Flussonic Media Server provides features to work with the archive, such as:

- Live-to-VOD
- Catch-up
- Timeshift, and so on.

Flussonic Media Server can work with cloud storages. It can upload files to the cloud storage and download them from it.

Flussonic Media Server also allows to export archive fragments as MP4 files.
For more information, see Export DVR segment to MP4 file and Download the DVR segment to MP4 or MPEG-TS file to a local computer.

To reduce the load on the storage and speed up the distribution of VOD content, set up SSD caching.

For more information about DVR and its features, see DVR.

Egress server

The egress server allows you to accomplish the following objectives:

- Provide a comfortable environment for people to communicate on the Internet in real-time video and audio conferences.
- Deliver the recordings of streams to the viewers.

Latency in video or audio is critical to a real-time live stream. Viewers may experience issues with audio and video, leaving them dissatisfied with the quality of the service. To provide a comfortable environment for real-time video and audio communication online, use WebRTC. WebRTC allows you to :

- Create a sense of live communication between the participants with latency less than a second.
- Automatically adjust the video quality to the speed of the Internet connection using the algorithm of adaptive bitrate (ABR). This makes it possible to expand the audience of private chats and for viewers with an unstable or slow Internet connection to watch the live stream.
- Watch the live stream on any device.
- No additional software required. All a viewer needs is a web browser and connection to the Internet.

Live streams with a delay that is enough to respond to the chat, but not enough to communicate with viewers live are most common. In such cases, a small latency isn't critical, so use HLS, DASH, and MSS protocols to deliver streams to viewers. It makes it possible to:

- Watch live streams on any device.
- Expand the audience that doesn't require ultra-low latency.
- Improve the effectiveness of the sales funnel.

Egress server captures the streams from one the following servers:

- the transcoding server
- the DVR server
- the publishing server

Egress server then delivers the stream to the viewers directly or with the CDN, depending on the objectives and the architecture design.

For more information, see Pushing a stream to other servers.

To show the streams to the viewers you can use *Flussonic* web player and embed it on your website. To integrate the player with your website, use the Flussonic Streaming API.

Central

Central helps to:

- Ensure the hardware is used under uneven load during the time of day.
- Ensure even distribution of network traffic among servers.

Central acts as a management server in *Flussonic* that provides a central point of access for the following tasks:

- Managing and coordinating server configurations for all the servers in a content delivery pipeline.
- Authentication and authorization of creators and viewers.

Central allows you to make a highly available server configurations for the servers in the video delivery pipeline. This means you don't have to configure each server manually. *Central* interacts with other servers using a built-in mechanism—config_external.

Central also handles load balancing between servers in the cluster, redirecting requests to the least loaded servers.

Flussonic provides the Flussonic Central API to communicate with *Central*.

You can implement the authorization using *Flussonic* built-in tools or with an external backend.

If you don't have a backend for authorizing creators and viewers or don't want to create one, you can use the *Flussonic* built-in tools, such as:

- Password-protected publishing stream: checks the password before publishing a stream.
- `passphrase` -protected SRT publishing: checks the password before publishing an SRT stream.
- Backend configurator: creates authorization backend in the *Flussonic* configuration file instead of scripts.
- `passphrase` -protected SRT playback: checks the password before playing an SRT stream.

If you already have an authorization backend, configure *Flussonic* to access it using these tools:

- Token-based authorization: accesses the specified backend and verifies the received token.
- External backend authorization of playback sessions: accesses the specified backend to check if a viewer has a permission to play the content.
- External backend authorization of publishing sessions: accesses the specified backend to check if a creator has a permission to publish the content.

Flussonic also provides the Flussonic Authorization Backend API to work with the authorization backend.

Find the information about the components and their features in the Table 1:

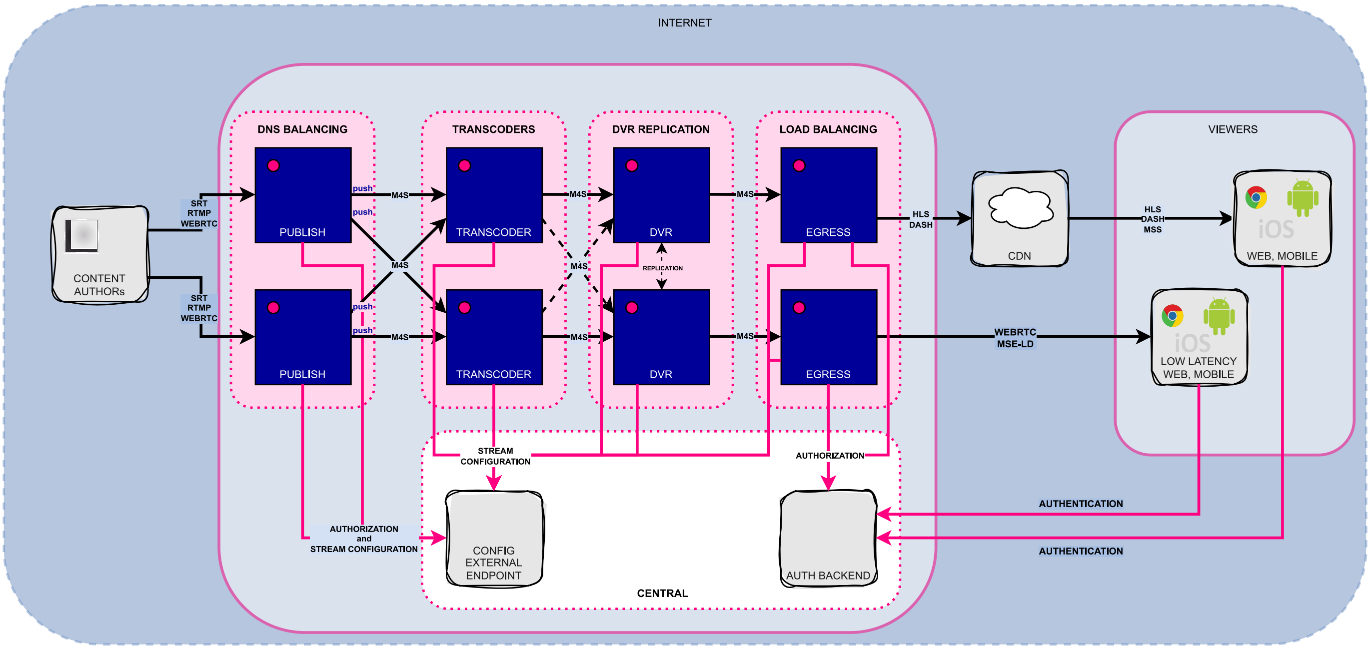
Table 1. Content delivery pipeline components and their features

Component	Features
Publishing server	- receiving streams - multistreaming to other streaming platforms and socials
Transcoding server	- changing audio and video parameters (codec, bitrate, resolution, etc.) - creating a multi-bitrate stream - overlaying logo
DVR server	- recording and storing copies of incoming streams - working with the archive and its features (live-to-VOD, timeshift, etc.) - playing streams from the archive (rewind, watching the current live stream from the beginning)
Egress server	delivering streams to viewers: transcoded streams (for live viewers) and copies of transcoded streams (for DVR users)
Central	- managing and coordinating server configurations for all the servers in a content delivery pipeline - authentication and authorization of creators and viewers - server load balancing

Workflow

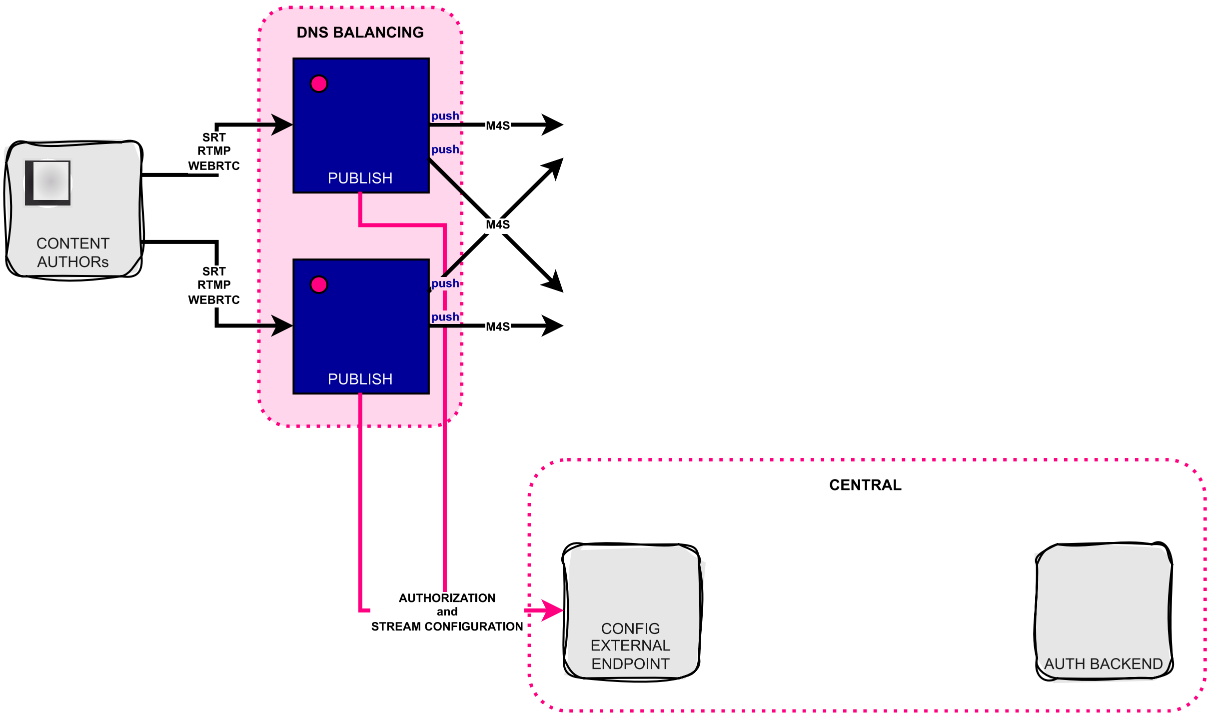
This chapter is devoted to the process of how the UGC streaming platform works, what steps the content goes through to get to the viewer's screen.

Get back to the diagram of a UGC streaming platform:



The black arrows in the diagram show the direction of streams, and the pink arrows indicate the direction of requests.

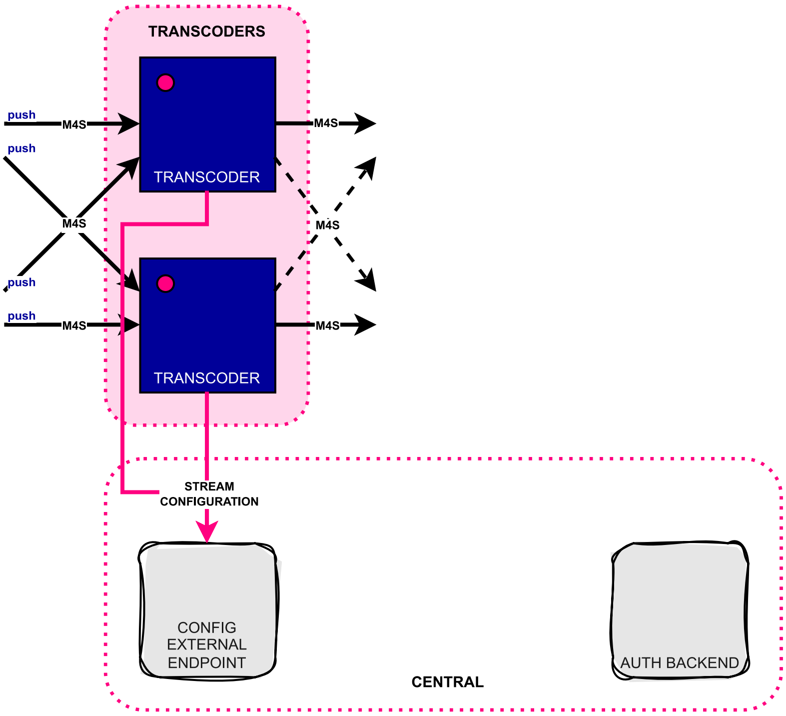
1. Creator <-> Publishing server <-> Central



To start streaming, the creator needs to authorize and initiate stream publishing to the publishing server via SRT, WebRTC (WHIP) or RTMP, depending on the source. Here is how the creator starts streaming:

1. The creator's request goes to the DNS server.
2. The DNS server redirects the request to the address of one of the available publishing servers in the cluster. See below for details on DNS balancing.
3. The publishing server sends a request to *Central* to authorize the creator.
4. Once the publishing server receives a positive response, it requests the configuration for the stream from *Central*.
5. *Central* returns the necessary configuration to the publishing server.
6. The creator publishes the stream.

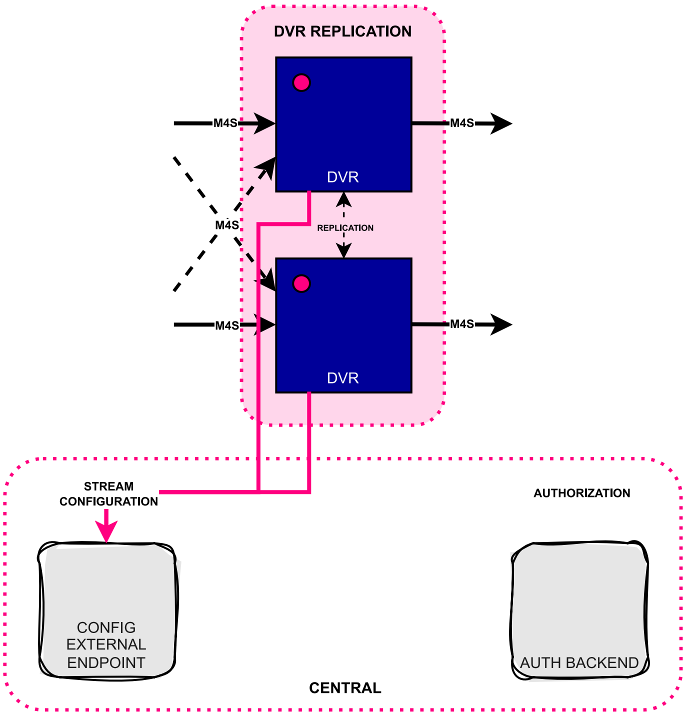
2. Publishing server -> Central -> Transcoding server



1. The publishing server repackages the incoming stream into Flussonic-to-Flussonic M4S protocol and sends it to a cluster of transcoding servers.
2. *Central* as a load balancer checks health states of all the servers in a cluster and redirects the stream to the least loaded server.
3. The transcoding server receives the M4S stream, unpacks and decodes it, getting the raw video and audio.
4. The transcoding server converts stream into a stream with multiple video tracks with different resolutions and bitrates.

> **Note**
>
> When receiving a stream, the *Flussonic Media Server* unpackages the stream and repackages it. It increases the stability of the output stream.

3. Transcoding server -> Central <-> DVR server

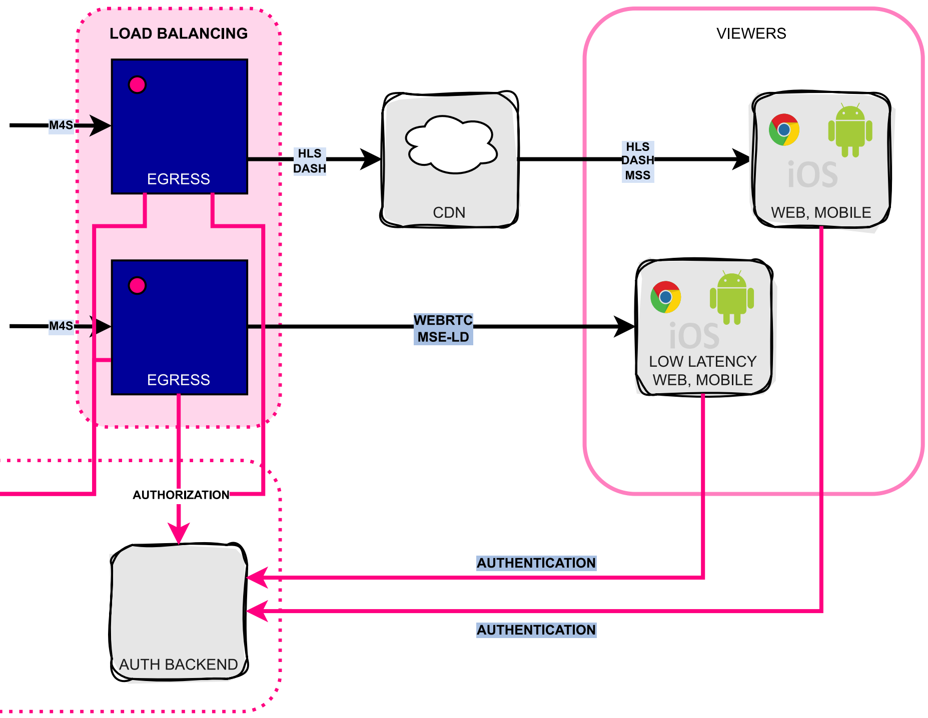


The DVR server accomplishes the following tasks:

1. Requests the stream configuration from *Central*.
2. Captures the M4S streams from the transcoding server, records them, and stores copies of those streams.

The DVR server also communicates with another DVR server in the cluster to replicate the archive. This way, the archive is replicated to other servers to ensure reliability and automatic data recovery after failures. See below for more information on backup.

4. DVR server <-> Central <-> Egress server <-> Viewers



Viewers need to be authenticated and authorized to watch the stream.

1. The viewers access the website from their devices and enter their usernames and passwords. An authentication request is then sent to the *Central* server.
2. *Central* checks the viewers and authenticates them.
3. After successful authentication, the viewer sends the request to the *Central* load balancer.
4. The *Central* redirects the request to the least loaded and available egress server in the cluster.
5. The egress server then sends a request to the *Central* server to authorize the viewer.
6. *Central* checks if the viewer has a permission to play the stream.
7. Based on the response received, the egress server either sends the stream to the viewer or not.

Streams are delivered in two different ways, depending on whether the stream is live or recorded:

1. From the egress servers directly to viewers.

   When viewers request a live stream through a browser or mobile device, stream is delivered via WebRTC and MSE-LD. WebRTC and MSE-LD provide lowest delay when playing the stream.
2. From the egress servers to CDN and then to vewers.

   When viewers request a recorded stream through a browser or mobile device, the stream is delivered via HLS and MPEG-DASH via CDN. CDN allows to deliver the content to the viewer quickly, reduce the network costs and optimize the load on servers.

Redundancy and scaling strategies

This section provides architecture design ways to build a reliable service. It's essential for a reliable service to meet these requirements:

- Keep responding to customer requests in spite of a high demand on the service or a maintenance event.
- Handle failures.
- Scale in response to changes in workloads and user demands.

Cluster of publishing servers and DNS balancing

The above diagram uses a cluster of several DNS-balanced publishing servers. This approach allows to:

- Distribute the load on several servers.
- Ensure the creator connects to at least one active server.

To provide a redundant service when publishing SRT and RTMP streams that aren't supported by the application layer, use DNS balancing. To distribute publishing WebRTC (WHIP) streams, use the built-in *Flussonic* load balancer.

All the publishing servers in the cluster are the same, i.e. they are equal and interchangeable, and each of them can accept any stream. So if one of the servers in the cluster fails, another server automatically takes over.

In DNS balancing, the server to which the request will be sent to is selected according to a certain algorithm, for example, round robin. There are several ways to find out the address of the server to send the creator's request to: request it via API (recommended) or return the appropriate one using network methods. It depends on the architecture of your network.

However, DNS has a significant disadvantage—it doesn't check the server load of the servers in the cluster. That is why creators may be redirected to a server which may be currently overloaded or down.

Also keep in mind that:

- DNS is cached for up to three days. If there are servers in the cache that are down or in a closed network, the connection timeout may be very long.
- RTMP has no built-in redirection mechanism. Temporary solutions made to fix this don't work. However, WHIP (WebRTC HTTP Ingest Protocol) is balanced by redirects and proper made SDP.

In our case, we need a balancer that redirects requests rather than a proxy. If one publishing server fails, creator requests will be redirected to remaining active servers.

Let's consider the use of DNS balancing on a cloud platform that provides UGC service functionality for various projects.

For a creator to connect to at least one active server, at least two DNS records are needed. So for each project a separate group of servers is provided, according to server load or, for example, the geographical location of the creators.

It provides the following features:

- Multiple DNS entries for redundancy.
- A separate domain for each project to balance requests for publishing streams and resource allocation.

Load-balanced transcoder cluster

If a transcoding server crashes, it can cause the whole service to fail. To avoid this, follow these recommendations:

- Use at least two transcoding servers as a cluster.
- Provide a failover, so that if one of the servers goes down, other servers can take its workload.
- Distribute the load evenly between the servers.

We suggest clustering with load balancing using *Central*.

A cluster of two or more transcoding servers can ensure uninterrupted operation of transcoders. If one of the transcoding servers fails, the other servers will be able to run processes on their side. Thus, the service will continue to run and keep working as expected. For more information about clustering, see Cluster.

To avoid overloading the transcoding servers, it is necessary to distribute requests to the cluster of transcoding servers evenly. *Central* load balances requests following these steps:

1. Evaluating the load status of transcoding servers in the cluster.
2. Directing requests to the least loaded servers.

For more information about *Central* and load balancing, see Flussonic load balancing.

DVR cluster with replication

A sudden outage, system failure, or any other issue can cause DVR data loss. To secure your service, you need to back up your DVR archive and copy it to other servers. We suggest using a cluster of two or more DVR servers with replication.

Replication allows to copy the recordings to other servers to accomplish the following tasks:

- Ensure reliability.
- Provide automatic recovery after a system failure or shutdown.

A DVR archive is stored on two or more server instances, where one is a primary server and the others are secondary servers. Replication works like so:

1. Primary server pulls the streams from the source and stores.
2. The secondary server pulls the streams from the primary server (see the diagram).

Flussonic provides other tools to back up a DVR archive and keep it available at all times, such as:

- Cross-replication
- Flussonic RAID
- Cache

CDN

With the increasing number of viewers and amount of output traffic, at some point one server becomes incapable of handling the delivery of streams. As the viewers from all over the world tune in, it becomes challenging to provide them with a good and reliable service. It makes it overwhelming for an egress server to serve more and more requests at a time.

CDN (Content Delivery Network) allows you to accomplish the following objectives:

- Increase the number of viewers without spending a lot of money on hardware.
- Distribute the load evenly between the servers.
- Make the content accessible for the viewers.

Having multiple servers distributed across various geographic locations, CDN provides these advantages:

- Shortens the distance between the edge server and the viewer.
- Minimizes latency.
- Allows quick access to the content for the viewer.
- Reduces the load on the egress server to keep it up and running.

To provide an efficient delivery of the content worldwide, CDN uses the following methods:

- Caches the content in different PoPs (points of presence) to deliver it directly to the viewer.
- Distributes the workload between multiple servers.

You can build your own CDN or use existing solutions, such as Akamai, CDNvideo, and others, to do the job. It depends on the number of viewers, their geographical location, allocated budget, and so on.

ANALYZE AND ESTABLISH REQUIREMENTS

When starting a project, the initial step is to perform an analysis and establish the requirements.

To establish the requirements, follow these steps:

1. Define a niche
2. Study your target audience
3. Define the content and its creators
4. Define the key features of the service, having studied the market.

Define a niche

Choose the kind of market area your service will serve to. Is it going to be health and fitness, online learning, arts and crafts, or gaming, and so on? Will it cover one use case or several ones?

Defining the niche is crucial for a UGC streaming service or platform to determine the objectives of your service or platform, identify the target audience, develop your marketing plan and further work. Research the market, see what your competitors have to offer.

Study your target audience

Define who your audience is and what they want. Answer the following questions:

- What do your viewers have in common?
- Why does the audience watch the content? For education purposes, entertainment, sport, an so on?
- What devices do viewers use to watch the content? PC, mobile phone (Android, iOS), and so on?
- What approximate number of concurrent viewers must the service or platform handle?

By answering these questions, you can identify your target audience and what they expect from your UGC streaming service or platform.

For example, the target audience of the gaming industry enjoys video games and watches walkthroughs, eSports competitions, an so on. Viewers use PCs, laptops, mobile phones (Android, iOS), tablets. It's an example of entertainment content watched by hundreds of thousands of viewers.

Define the content and its creators

Determine what content you want to provide and who will be providing this content. Determine what your target audience needs and their preferences to define what content they will watch.

Consider *Twitch*. *Twitch* was initially created as a live streaming platform for gamers. Gamers stream video game walkthroughs and gameplays, tournament organizers stream eSports competitions.

When determining what content to provide through your platform or service, rely on the target audience and its preferences.

Define the key features of the service

When creating a streaming service or a platform, provide features for your customers that meet their needs and demands.

Here are some of the common features of UGC streaming platforms:

- for creators: stream with different tools, for example, VMix, OBS, Atemi, HDMI-RTMP converters, video editing consoles, and web browsers
- for viewers: watch the content from any device, for example, PC, mobile phone, laptop
- record, store and play live streams
- multistream to social media and platforms, like YouTube, Instagram, Facebook, Twitter, Twitch
- continuous and stable playback of content
- upload pre-recorded streams
- low-latency streaming
- embedded media player
- authorization system
- subscription system, and so on

Depending on the goals and objectives of the platform or service, allocated budget, and available hardware and software resources, the set of features can vary.

DESIGN THE NETWORK ARCHITECTURE

This chapter describes how to do the following tasks:

- Estimate the required input and output network bandwidths for a publishing server, a transcoder, a DVR, an egress server.
- Create a network architecture diagram.

To calculate the input and output bandwidth and create a network architecture diagram, answer these questions:

Table 2. Questions to determine the network requirements

Topic	Questions
Source (input)	1) What is the type of signal source (camera, software encoder, hardware encoder, browser)? 2) What are the parameters of the input stream (codec, bitrate, resolution, FPS)? 3) Which streaming protocol is used for transmitting video from the source (RTMP, SRT, WebRTC)? 4) What is the total number of incoming streams?
Transcoder	1) What are the output video stream profiles (codec, bitrate, resolution)? 2) What is the total number of output streams for distribution? 3) Is audio transcoding required? If yes, what are the output audio stream profiles (codec, bitrate)? (For example, Opus (WebRTC) -> AAC (HLS, DASH))
DVR archive (recording and storing)	1) Is stream recording required? If yes, how do you plan to use DVR (export recordings to VOD, implement catch-up, etc.)? 2) What is the required depth of the archive (one day, week, month)? 3) Where will streams be stored (cloud storage, disk, RAID)?
Egress server (output)	1) What are the types of client devices (mobile phones, PCs, and so on)? 2) What is the streaming protocol (HLS, MPEG-DASH, WebRTC, etc.) for content distribution? 3) What is the estimated number of viewers? 4) Are you planning to distribute content using your own resources or third-party CDN (Akamai or others)?
Security	1) Is authorization required? If yes, by what means: external backend or built-in *Flussonic* tools? 2) Is content encryption necessary?
Other requirements	1) Are audio tracks with multiple audio tracks with different languages? 2) Do you need additional control of any video parameters? and so on.

To calculate the network bandwidth, use the following formula:

*number of streams * (video bitrate + audio bitrate)*

Based on the questions from *Table 2*, you can determine the number of publishing servers, transcoders, DVR servers, and egress servers and create a diagram.

Then move on to setting up and configuring the servers.

EXAMPLE OF AN IMPLEMENTATION STRATEGY FOR A UGC SERVICE

This chapter shows how to develop an implementation strategy for an event broadcasting platform. Follow these steps:

1. Analyze and establish requirements
2. Design the network architecture diagram

> **Warning**
>
> We don't provide any programming code or server configuration for this project.

Analyze and establish requirements

Suppose, you want to create an event broadcasting platform.
The project goal is to reduce company's expenses on the current service.

- Niche: event broadcasting
- Target audience: big and small companies
- Type of content: summits, webinars, online workshops, online corporate events, conferences
- Key features:
    - Support VMix, OBS.
    - Record live streams and store them, so the viewers can watch them later.
    - Provide failover and backup to keep the service up and running if any issues occur.
    - Ensure that viewers with slow internet connection can watch the stream.
    - Manage the available resources efficiently at uneven server loads.
    - Make the content accessible from any mobile devices and browsers.
    - Provide statistics of a system performance.
    - Authenticate and authorize viewers and creators.

Design the network architecture

You gave the following answers to the questions from Table 2:

Topic	Description
Source (input)	1) Source type: a hardware encoder or a software encoder like vMix. 2) Stream parameters: H.264, 5 000 Kbps, 1920x1080p, 25 FPS. 3) Streaming protocols: RTMP or SRT. 4) Number of input streams: up to 10.
Transcoder	1) Transcoding in three profiles: Full HD (1080p), HD (720p), SD (360p). 2) Number of output streams: up to 30. 3) Audio transcoding isn't required.
DVR archive (recording and storing)	1) Recording and storing of streams is required. Viewers will have the opportunity to watch an event wherever and whenever they choose and to download the recording to their device. 2) Up to 10 hours for all 10 streams in a separate data center. It's neccessary to be able to export the archive fragments as MP4 files. 3) S3 cloud storage stores the recorded events.
Egress server (output)	1) Client devices: PC and mobile phones. 2) Streaming protocol: HLS. 3) Estimated number of viewers: up to 100,000. 4) Using CDN Akamai to deliver the content.
Security	1) Authorization of users is required. 2) Encryption isn't neccessary.
Other requirements	1) To create streams automatically, API integration is required.

To calculate the network bandwidth, we use the formula:

*number of streams * (video bitrate + audio bitrate)*

- The input and output bandwidths for the publishing server:

  *10 streams * (5,000 + 192)Kbps ≈ 52 Mbps*

- The output bandwidth for the transcoder. Transcoding ten Full HD (1080p, 5,000 Kbps, H.264) streams into three video profiles each:
  - Full HD (1080p), H.264, 4,000 Kbps; AAC, 192 Kbps
  - HD (720p), H.264, 2,000 Kbps; AAC, 128 Kbps
  - SD (360p), H.264, 1,000 Kbps; AAC, 96 Kbps

  Makes:

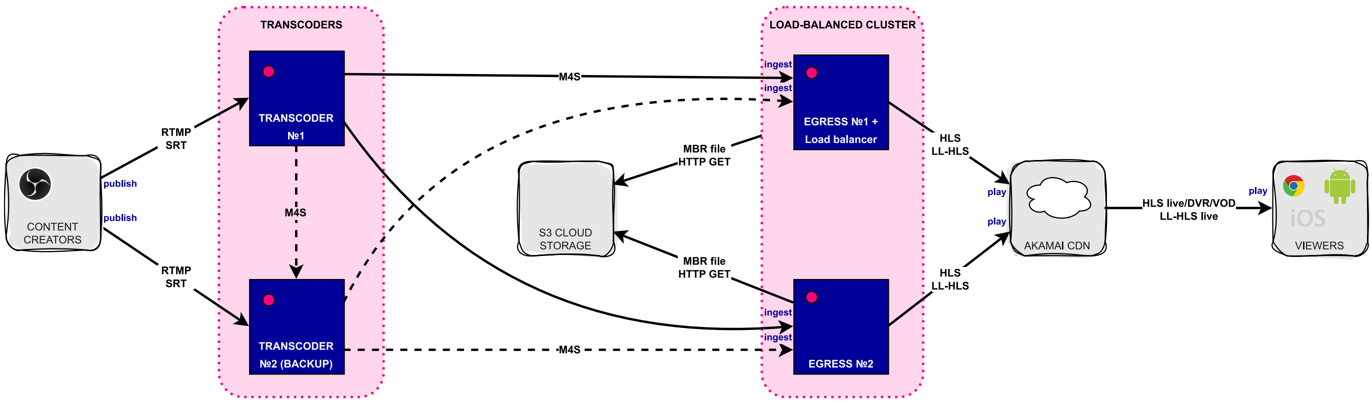
  *30 * (4,000 + 2,000 + 1,000 + 192 + 128 + 96)Kbps ≈ 223 Mbps*

- The input and output bandwidths for the DVR equal the output bandwidth for the transcoder.
- The output bandwidth for the egress server, given that all 100,000 viewers watch the Full HD stream at the same time:

  *100,000 * (4,000 + 192)Kbps ≈ 420 Gbps*

Now we can make a network diagram:

Diagram 3. Network architecture diagram for an event broadcasting platform



Four *Flussonic* servers are required:

- Transcoding server №1 and transcoding server №2: receive and transcode the streams.
- Egress server №1 and egress server №2: record the streams and deliver the content to the viewers.

Authorization

The platform authorizes viewers by their specific token:

1. A website generates a unique token for each viewer so that the viewer cannot pass it on to others.
2. *Flussonic* checks the token, the IP address and some other parameters.

This methos is great for private events.

Transcoding servers

Transcoding server №1 and №2 receive RTMP and SRT streams and transcode them.

Transcoding server №1 is the main server, and transcoding server №2 is the standby one. This way we can cover these scenarios:

- If the main server becomes unavailable, the standby server takes over.
- If both main and standby servers become unavailable, the fallback video file will start on a loop until one of the servers goes back online. The fallback video will also be written to the archive.

Egress servers

Egress server №1 and egress server №2 accomplish these tasks:

- Record live events and write them to the archive.
- Deliver the streams to viewers.

To record and store the live streams, the system works as follows:

1. Egress server №1 and egress server №2 capture M4S streams from the transcoding server and record them.
2. After the live event is finished, *Flussonic* automatically uploads the recordings as MP4 files to S3 cloud storage.

To speed up the delivery of VOD, we enable SSD caching. Here is how it works:

1. The first time an MP4 file with a live stream recording is requested, the file will be cached locally on the SSD.
2. All subsequent requests will be served from this SSD.
3. If the cached file isn't accessed for 24 hours, then *Flussonic* removes the file from cache.
4. If the total size of files stored in cache exceeds 100 GB, then *Flussonic* deletes files from the cache, starting with the oldest one.

To distribute the load between the servers, we use a hybrid approach:

- Load balancer on the side of CDN Akamai handles play requests for large public events. CDN Akamai captures the HLS and LL-HLS streams from the egress server №1 and egress server №2.
- *Flussonic* load balancer distributes the load between the two egress servers for small events. In this case, egress server №1 serves as a load balancer.

To deliver the streams to viewers we use HLS and LL-HLS streaming protocols:

- LL-HLS for live streams
- HLS for VOD

GLOSSARY

User-generated Content (UGC)

any form of content, for example, videos and images, created by people and published online.

UGC platform

A software-as-a-service (SaaS) solution used to collect and manage the content. It connects a creator and a viewer through content. A crucial feature of a UGC platform is that it uses content creators to make the content, when IPTV/OTT platforms are content providers themselves.

Publishing server

A server that receives a stream from the creator.

Transcoding server

A server that transcodes a stream into an array of streams at different resolutions and bitrates.

DVR server

A server that takes incoming video streams, records and stores them, and then sends them on demand.

Egress server

A server that delivers the content to the viewers.

Cluster

A group of servers working together to perform common tasks.

DNS balancing

A method of assigning several IP addresses to the same domain name, which allows you to direct creator requests to different servers when accessing the same domain.

VOD playback

Content:

EMBEDDING THE VIDEO PLAYER IN A WEBPAGE

Flussonic's built-in HTTP server can serve a special **embed.html** page to client software. This page can be used to insert video into a website, or to view VOD content in a browser.

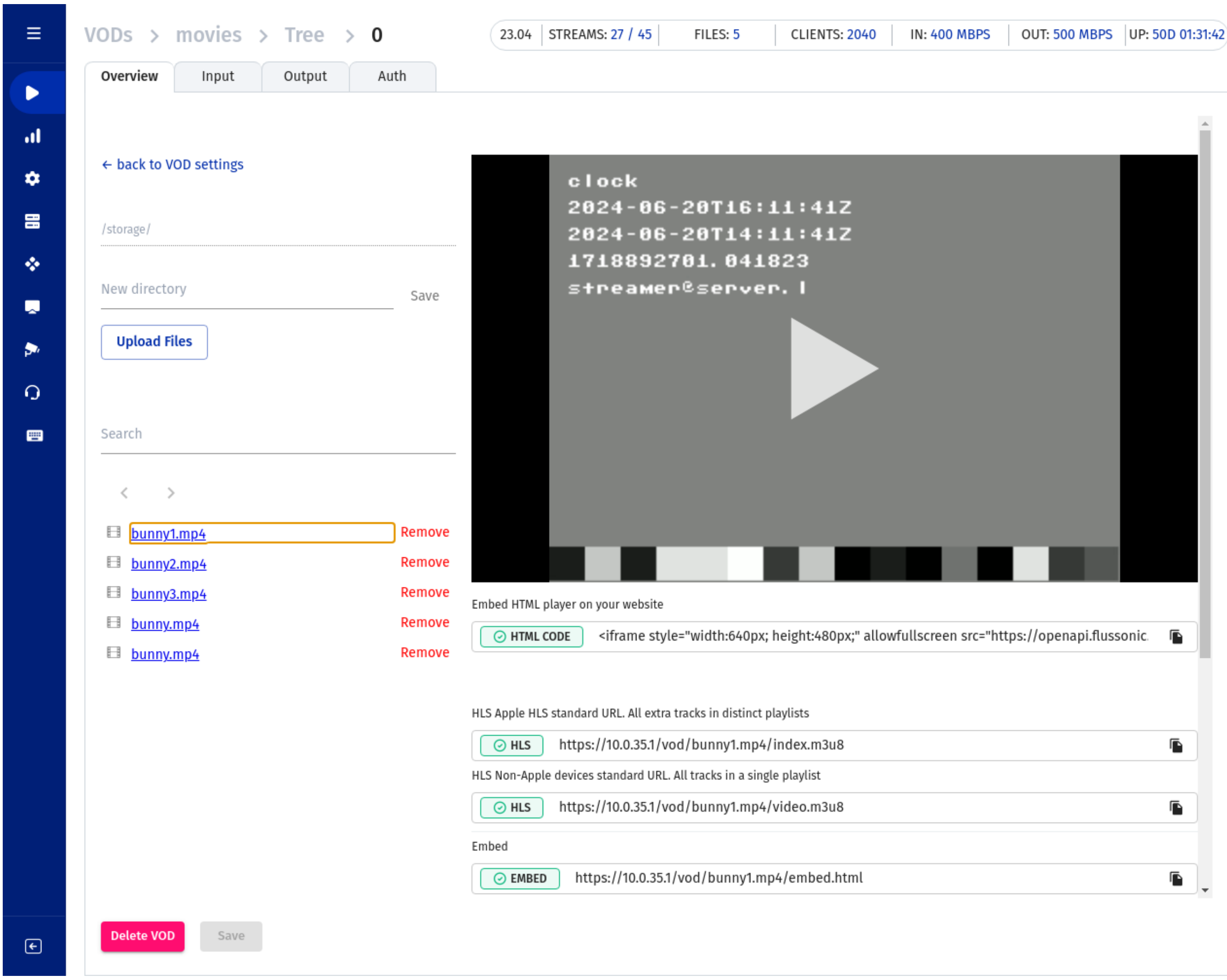
It is available using the following URL:

```
http://FLUSSONIC-IP/myvod/bunny.mp4/embed.html
```

Learn more about this feature in this article: Adding video to websites (embed.html).

PLAYING FILES OVER DIFFERENT PROTOCOLS

This section demonstrates how to play VOD files over several different protocols. You can see the list of all supported protocols along with the URLs for playing in the web interface. You can also play the VOD files here. Just go to **Media** — **VODs** — your VOD — **browse** and select the file. The player and the list of URLs is displayed on the right:



> **Note**
>
> Note that the UI can only play files that are in VOD locations.

You can also make the URL for playing VOD manually. Below you will find some examples of play URLs for a file at the path `/movies/example/s01e02.mp4`. Let's assume that we have configured a VOD location like this:

```
vod myvod {
  storage /movies;
}
```

For the file on disk in `/movies/example/s01e02.mp4`, you should use the following URLs as sources for the player:

- **Playing a VOD file via HLS (iOS, Android, STB)**

```
http://FLUSSONIC-IP:80/myvod/example/s01e02.mp4/index.m3u8
```

- **Playing a VOD file via MSS**

```
http://FLUSSONIC-IP:80/myvod/example/s01e02.mp4.isml/manifest
```

- **Playing a VOD file via DASH**

```
https://FLUSSONIC-IP:80/myvod/example/s01e02.mp4/index.mpd
```

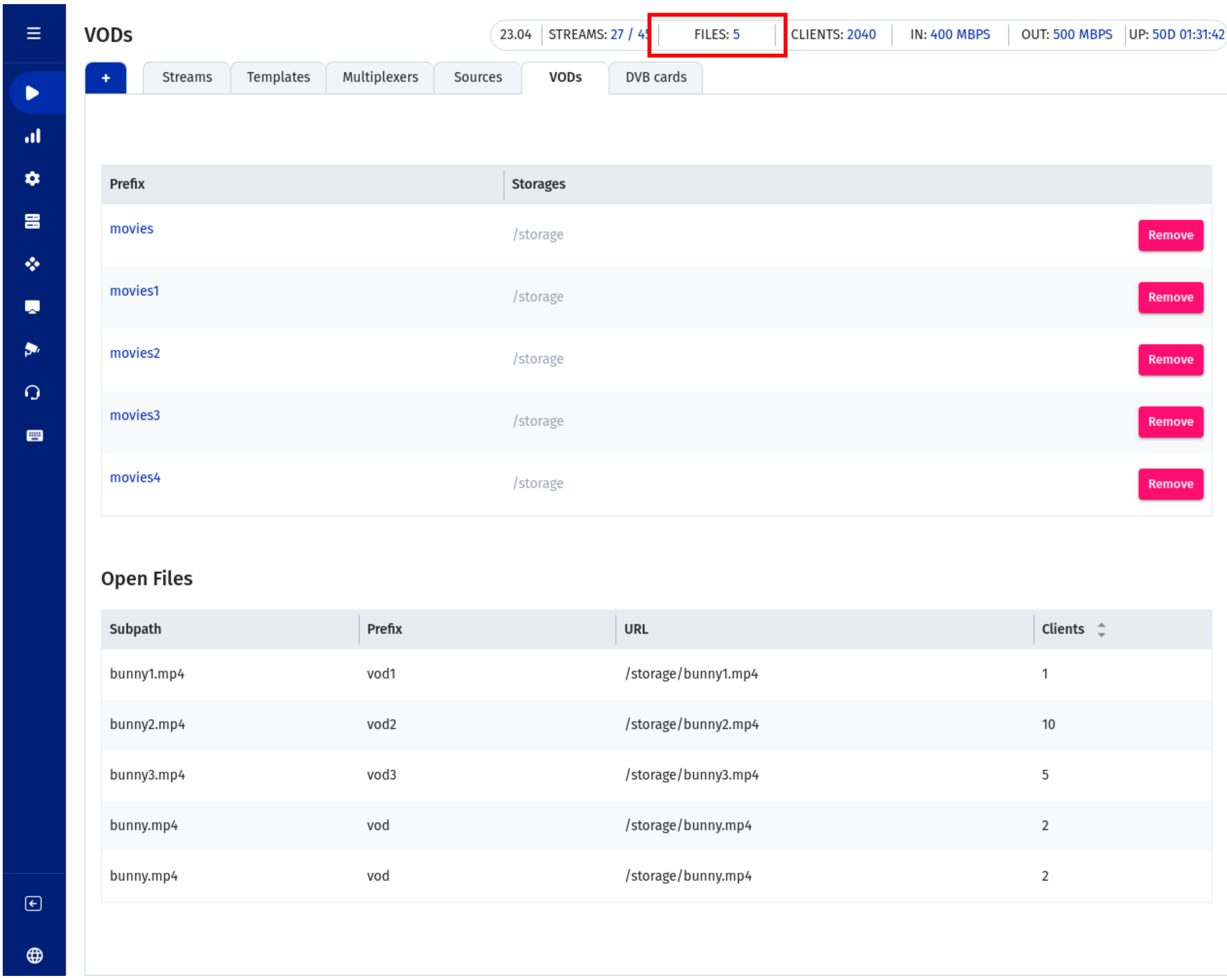
- **Playing a VOD file via RTSP**

```
rtsp://FLUSSONIC-IP:80/myvod/example/s01e02.mp4
```

> **Note**
>
> Set up the RTSP port in **Config** —> **Settings** —> **Protocols** to use RTSP.

MONITORING THE VOD FILES PLAYBACK

Statistics on files that are currently open is displayed on the **Media** —> **VODs**. Also, the number of open files is indicated on the information panel in the upper right corner of the screen.



MULTI-LANGUAGE BROADCASTING

HLS protocol allows broadcasting video content with multiple audio and subtitle tracks, each in a different language. Flussonic Media Server automatically enables multi-language broadcasting if extra audio tracks or tx3g subtitle tracks are detected in a VOD file.

EXPORTING SUBTITLE TRACKS AS SRT

Flussonic Media Server can serve subtitle tracks in the SRT (SubRip Text) format. This is the only way to display subtitles on certain platforms (e.g. some Flash players). A VOD file's SRT subtitle track can be retrieved over HTTP:

```
http://FLUSSONIC-IP:80/myvod/video.mp4/track-t1.srt
```

ADAPTIVE STREAMING (MULTI-BITRATE)

Adaptive streaming can be used to make sure that clients with low-bandwidth connections have a good viewing experience. There are two ways to enable adaptive streaming in Flussonic Media Server:

- Use several files with the same content but with different bitrates.

  You can configure *Flussonic* to automatically create a single multi-bitrate playlist, or use the SMIL files.

- Use a single file that contains multi-bitrate content.

  To play back a multi-bitrate MP4 file, you will need to create a manifest file for it. The Preparing Multi-bitrate Files section of our documentation gives detailed instructions on creating multi-bitrate files.

RESTREAMING VOD FILES

Copying a large VOD library between servers can be expensive. Flussonic is able to re-stream video files from one Flussonic server to another. This saves not only time but also storage space required to store VOD content. Saved resources can be used to enable caching of VOD content, which will increase the performance of a VOD re-streamer.

Example of VOD location configuration:

- config file on the source VOD server:

```
vod source_vod {
  storage /storage;
  download;
}
```

- config file on the restreamer VOD server:

```
vod restream_vod {
  storage http://FLUSSONIC-IP:8081/source_vod;
  cache /mount/cache 500G misses=2;
}
```

2.3.3 API

Flussonic API list

Welcome to our API references section!

All the API References currently available for *Flussonic* technologies are listed in this section.

We provide the following public API references:

- Flussonic Media Server API
- Streaming API
- Flussonic Central API
- API for authorization backend
- API for config_external
- Watcher Client API
- Watcher Admin API
- API for custom stream layout manager

Flussonic API design principles

Table of contents:

GENERAL INFORMATION ABOUT THE API DESIGN

We at Flussonic provide various products and services, such as *Media Server*, *Watcher*, *Iris*, *Retroview* and so on.

This page provides up-to-date information on the HTTP API management principles concerning our systems.

PRINCIPLES OF API DESIGN

The Flussonic HTTP API is designed for other applications to access our systems and interact with them. Here are the core principles that we try to follow:

- Focus on industry standards and generally accepted practices while maintaining the principle of reasonableness.
- Organizing the API around REST + JSON.
- Using OpenAPI 3.0 format (formerly Swagger) for a machine-readable API description.
- API-first approach. This means that we design the API before generating the code based on it using the openapi_handler that we developed to support the approach and made available to the public.
- Strict compliance with the API schema returned in the payload data. No fields that are not in the schema can be received from the service.
- Flexible compliance with the API schema to the data sent in payload. Attempting to send a known field in an incorrect format will result in an error. Unknown fields in the payload will be ignored.
- Convenience and simplicity in creating both simple and complex requests (for example, a simple request for streams list and a complex request with multiple filters).
- Consistency in terminology among the systems (the name of an entity in one system must be the same among all the others).
- Standardized access methods to different systems. There might be a collection of one authorization backend on the server and millions of session records in the browsing history storage service. Standardized API should provide access to such systems.
- Preference for HTTP. Access to data on WebSockets is poorly standardized and hard to monitor.
- Obtaining a reasonably limited amount of data for each sample. So the client will not receive a big amount of records by default.
- Supporting the work with a dynamic data type.
- Support of GET requests and declared parameters in the query string to create simple access requests to large collections.
- Idempotent API, i. e. running the same request multiple times produces the same result.
- Supporting work with complex objects, nested subobjects, and collections.

AUTHENTICATION AND AUTHORIZATION

With **Flussonic API**, you can retrieve data and manage certain *Flussonic* features over HTTP.

To enable the protection for **data retrieval** requests use the `view_auth user password;` directive, and for **modification of a state and settings** requests with the `edit_auth user password;` directive in the configuration file `/etc/flussonic/flussonic.conf`.

For more information about the authorization in *Flussonic* see: Authorization.

To get an access to Flussonic HTTP API functionality with enabled authentication use a login and password according to HTTP Basic Auth.

Authorization for the `edit_auth user password;` looks as follows:

```
GET /streamer/api/v3/streams HTTP/1.1
Host: FLUSSONIC-IP
Authorization: Basic dXNlcjpwYXNzd29yZA==

HTTP 200 OK
Date: Sun, 19 Sep 2021 19:40:22 GMT
Content-Type: application/json
...
```

If you need to use `Bearer` auth, use:

```
GET /streamer/api/v3/streams HTTP/1.1
Host: FLUSSONIC-IP
Authorization: Bearer dXNlcjpwYXNzd29yZA==

HTTP 200 OK
Date: Sun, 19 Sep 2021 19:40:22 GMT
Content-Type: application/json
...
```

> **Note**
>
> Authorization used in other systems may differ, but, in most cases, `Bearer` auth is used.

OPENAPI SUPPORT

Systems that support the principles mentioned in this document provide service descriptions according to the machine-readable OpenAPI 3.1 specification, which evolved from *Swagger* and *JSON Schema*.

OpenAPI defines a standard, language-agnostic interface, which allows formalizing a list of service methods and the input and output data formats.

To retrieve the Flussonic API schema, use the following command provided that your *Flussonic* server is up and running:

```
curl http://FLUSSONIC-IP:8080/streamer/api/v3/schema
```

We also provide public API references as listed on the API overview page. You can get the schema from any of those API Reference URLs by adding `.json` in the end, for example:

https://flussonic.com/doc/api/reference.json

The example of code below shows the way you can access a list of streams with *React Query* and *openapi-client-axios*:

```
import React from 'react';
import { useQuery } from 'react-query'
import OpenAPIClientAxios from 'openapi-client-axios';


const api = new OpenAPIClientAxios({
  definition: 'http://localhost:8080/streamer/api/v3/schema',
  axiosConfigDefaults: {
    headers: {
      'Authorization': 'Basic dXNlcjpwYXNzd29yZA==',
    },
  },
});


function Streams() {
  const { isLoading, error, data } = useQuery({
    queryKey: "streams",
    queryFn : () => {
```

```
      return api.init()
        .then(client => client.streams_list({}))
        .then(res => res.data);
    },
    keepPreviousData: true,
    refetchInterval: refetchInterval,
  });

  if(isLoading) return <div>Loading</div>;
  return <div>
    {data.streams.map((stream) => <div key={stream.name}>{stream.name}</div>)}
  </div>;
}
```

The program above reads the API schema, retrieves a list of *endpoints* and creates a set of functions from it.

In the example above, `streams_list` is taken from the schema and created programmatically.

Public and private API

The response to a request may return fields that are not described in the schema. Such fields are part of the **private** API. Ignore them like, for example, deprecated fields.

We use the private API to introduce experimental fields that can change without notice or backwards compatibility. When we are confident that such experimental field works as expected, you will see its description in the **public** API.

The request body may also include private parameters. Other fields that do not relate to either the private or public API will be ignored by Flussonic.

Deprecated fields

Some fields in Flussonic API are marked as `deprecated: true`. This means that we decided to get rid of this field after some time and use some new field instead.

Deprecated fields usually have `x-delete-at` parameter where we specify Flussonic version in which we are planning to delete this field. For example, `x-delete-at: 23.02` means that the field is planned for deletion in Flussonic version 23.02. If you are using this field, and don't want it to be deleted, you can contact us before the specified version and discuss leaving the field.

COLLECTIONS

Almost all objects are organized in **collections** (similar to SQL tables). The network's access to the system to read the data is mostly determined by the way objects are read from the collection.

For the fast-running user interfaces and predictably running programs, it is necessary to:

- be able to get the minimum required data set,
- be able to get all the data from the collection with certain reliability.

These two requirements are conflicting. The requirement to limit the data set implies getting only a **certain number** of objects from the collection, for example, 50 streams out of 2000 on the server.

When the client needs to get **the entire** list of streams, he will have to make **several** requests to the server. There is a chance that when new streams appear during the two consecutive requests, these new streams **will be missed** in the selection.

> **Note**
>
> We do not offer a general approach to capturing snapshots of the collection and allow a certain amount of risk of losing several records when performing page-oriented sampling from a dynamically changing collection.

For predictable access to a subset of the collection, we describe and implement a language that defines the following actions on the collection:

- filtering (similar to SQL `WHERE`),
- sorting (similar to SQL `ORDER BY`),
- limiting the number of elements (similar to SQL `LIMIT`),
- additional cursor filtering (similar to SQL `OFFSET`),
- limiting the set of fields (similar to SQL `SELECT`)

HTTP methods for accessing collections

The standard way to request a reading from a client to a server is using the query string language and the 'GET' method. It is considered that access that does not involve any modifications is carried out using a `GET` request (including caching).

> **Note**
>
> The caching is outdated and irrelevant in its original form, but we have implemented it for a convenient start.

When using a query string language, there are two approaches: to use a standard `key=value` approach or use non-standard delimiters. The second case may cause a number of challenges:

1. Too complex queries with nested conditions. Writing such conditions in a query string will look extremely clumsy and heavy, which contradicts our core principle — the convenience and simplicity of creating requests.
2. Incorrect query processing when using additional separators in subsets. In this case, it is necessary to remember the set of possible characters. For example, comma (",") or hyphen ("-") are not used in field names and are not processed in the query string. The ampersand ("&"), in contrast, is a special character for the query string and, if it is not escaped, issues may arise when the request is to be processed.

For example, some companies use the characters `-` and `+` in the sorting request to indicate the sorting direction. The `+` character is a special character and is treated as space when decoding a query string. Therefore, you either need to escape it using `%2B`, or use a non-standard parser to avoid possible issues with query execution. The use of non-standard parsers makes it difficult for standard libraries to create a request.

> **Note**
>
> There is a feature related to the use of Unicode characters. Some companies encode the SQL query such as `WHERE age > 20` in the query string as `age>20`. Using a Unicode character instead of the standard ASCII character `>` helps to avoid escaping to place into HTML document. However, it is challenging to type such a request from the keyboard. We refused to use Unicode characters, so it will not be possible to type them from the terminal.

The problems described above intend to illustrate the complexity of using a variety of conditions and versions of the SQL language into a limited *HTTP query string* language (query string language).

A more complex option is to transmit the request language in JSON format in the body of the POST request. This approach looks like an inevitable solution when using complex nested conditions.

Response structure

When **accessing the collection** *Flussonic* returns a JSON-encoded response with the following fields:

```
{
  "ITEMS": [...],
  "next": ...,
  "prev": ...,
  "estimated_count": ...,
  "timing": ...
}
```

`ITEMS` field is replaced with the name of the requested collection, i. e. `streams` for streams and `sessions` for sessions.

- `next` and `prev` are the cursor values (the next and the previous, respectively).
- `estimated_count` is an approximate number of elements in a collection.
- `timing` is a service object with various access times. It is not described as it may be changed or deleted in the future.

Filtering collections

To retrieve filtered responses, add the parameters in a URL query string.

Filtering by a value

To **filter by a value** (for example, `provider`), run the following command:

```
curl http://FLUSSONIC-IP:8080/streamer/api/v3/streams?provider=Sky
```

To check if **elements are in the list**:

`provider=Sky,Canal,CNN` (values are separated by a comma)

To impose a **condition on the nested object field**:

`stats.alive=true`

Filtering by a condition

It is also possible to use conditions for comparing field values during filtering. We decided to use special suffixes (like `_lt`, `_gt`, and so on) for coding such conditions. For example, the condition `WHERE stats.delay < 5000` turns into `stats.delay_lt=5000`.

We have defined a fixed set of suffixes, providing two non-overlapping sets with the existing set of field names among all our systems. It allows us to provide features to create access requests.

> **Note**
>
> For instance, the developers of one of the API variants decided to make not by direct comparison requests using `.`: `delay.lt=5000`. However, in this case, they cannot provide access to the fields of nested objects.

Here is a list of supported suffixes:

Query string parameter	SQL	Comment
`age=50`	`age = 50`	
`age_lt=50`	`age < 50`	
`age_lte=50`	`age <= 50`	
`age_gt=50`	`age > 50`	
`age_gte=50`	`age >= 50`	
`age_is=null`	`age IS NULL`	For comparison with NULL only
`age_is_not=null`	`age IS NOT NULL`	For comparison with NULL only
`age_like=pattern`	`age LIKE '%pattern%'`	For string parameters only

Several filters specified in the query string are applied *sequentially* to the collection, similar to `AND` queries in SQL:

```
curl  http://FLUSSONIC-IP:8080/streamer/api/v3/streams?stats.bitrate_gt=4000&stats.clients_count_lt=10
```

> **Note**
>
> We do not offer an `OR` version for query string due to the complex notation of parentheses.

> **Note**
>
> The filtering fields are specified without a prefix. For example, you could use them like the following `filter.age=50`. It may be more suitable for programming, validation, etc., but not for a person who will use such an API.

Sorting collections

Collection sorting is needed for displaying in the web interface *Flussonic UI* and retrieving a predictable page-oriented selection from the collection.

The sorting parameters passed separated by a comma to the key:

`sort=stats.ts_delay,-stats.bitrate`

By default, ascending-order sort is used. To change the sorting direction, specify `-` before the field name.

The absence of a prefix before the filtering fields means that we do not allow `sort` fields in objects in our API.

We added an **implicit** sorting everywhere in our API. For example, if sorting by the `provider` field (i. e. a non-unique field) is applied, then several groups of objects will be returned incomprehensibly sorted. To avoid such a situation, we add fields like `name` and `position` to the end of the sorting request, applying **implicit** sorting.

If they are specified explicitly in the list of sorting fields, then re-sorting by them is no longer performed.

We do not specify a list of specific fields for implicit sorting since it may change at any time if we decide to use it differently for a particular case.

Limiting collection rows (cursors)

If a collection is large, HTTP API should be able to return them part by part.

To get the first 100 rows (elements) of a collection, use the `limit=100` parameter in the query string.

The question is how to retrieve the next 100 elements.

> **Note**
>
> A solution for SQL databases is to pass the `offset` field. However, we refused to use such a method because it is computationally expensive (due to the Schlemiel the Painter's algorithm), and it is of no practical use. As we have already mentioned, the collections data may change between the consecutive requests, which means that the `offset` parameter will return the records not sequentially, but rather in an unknown order without any chance to determine what was lost.

> **Note**
>
> With page-oriented access, you do not need to get the *data with an offset of 100*. It is necessary to get *the next batch of data*. The approach with cursors is unusual for the `MySQL` due to their practical absence in this database. However, in older databases cursors are still in use.

In response to the request for the first 100 elements, the `next` field (and in the next samples `prev`) is returned. This field value should be passed in the `cursor=` parameter in the query string to fetch the following:

```
$ curl -sS "http://FLUSSONIC-IP:8080/streamer/api/v3/streams?select=name&limit=1&name_like=a&sort=name"  |jq
{
  "estimated_count": 5,
  "next": "JTI0cG9zaXRpb25fZ3Q9MiZuYW1lX2d0PWEx",
  "prev": null,
  "streams": [
    {
      "effective": {
        "name": "a1",
```

```
        "position": 2,
        "section": "stream",
        "static": true
      },
      "name": "a1"
    }
  ],
  "timing": {
    "filter": 0,
    "limit": 0,
    "load": 3,
    "select": 0,
    "sort": 0
  }
}
```

```
$ curl -sS "http://FLUSSONIC-IP:8080/streamer/api/v3/streams?select=name&limit=1&name_like=a&sort=name&cursor=JTI0cG9zaXRpb25fZ3Q9MiZuYW1lX2d0PWEx"  |jq
{
  "estimated_count": 5,
  "next": "JTI0cG9zaXRpb25fZ3Q9MyZuYW1lX2d0PWEy",
  "prev": "JTI0cG9zaXRpb25fbHQ9MyZuYW1lX2x0PWEyJiUyNHJldmVyc2VkPXRydWU=",
  "streams": [
    {
      "effective": {
        "name": "a2",
        "position": 3,
        "section": "stream",
        "static": true
      },
      "name": "a2"
    }
  ],
  "timing": {
    "filter": 0,
    "limit": 0,
    "load": 1,
    "select": 0,
    "sort": 0
  }
}
```

Using consecutive queries, we get **all** the elements of the data selection. The cursor is arranged quite simple: the values of the sorting fields for the last element of the selection are encoded in it, and an additional filtering is performed before the data is returned.

Limiting the field set of the result

We provide the possibility to return only a limited set of fields. The total number of fields in the stream data of the media server exceeds 100, and if you need to get only the stream names, then it is rather impractical to request all of the information.

To return only a limited set of fields, specify the `select` option, for example, `select=name,title`. You can also request the fields of nested objects: `select=name,status.media_info`.

CREATING AND UPDATING (UPSERT)

> **Warning**
>
> We are conforming Flussonic Media Server API to the JSON Merge Patch standard. It will let us bring the method of partial list update to a common format among all products of the *Flussonic* ecosystem. For now, we recommend that you start passing entire nested lists in API requests.

The REST concept dictates the distinction between creating and updating methods for already existing objects.

We have practically abandoned this separation and prefer a more robust option with simultaneous creation or updating of objects. If the methods for creating and updating are separated, the executed code has to have a condition expression with repetitions and perform a loop check. We made it so that, firstly, an object is created on the server. If we get a response that such an object already exists, then try to update it (or not), depending on the situation.

In fact, this concept corresponds to JSON merge patch document format processing rules:

- If the provided merge patch contains JSON members that do not appear within the target, those members are added.
- If the target does contain the member, the value is replaced.
- Null `null` values in the merge patch are given special meaning to indicate the removal of existing values in the target.
- If the patch is anything other than an object, the result will always be to replace the entire target with the entire patch.
- It's not possible to patch part of a target that is not an object, such as to replace just some of the values in an array.

> **Note**
>
> You don't need to resend required fields if they already exist in the target and haven't changed.

> **Note**
>
> If the *idempotency token* is not implemented on the server and the client, i. e. unique action ID, then repeated requests on object creation can lead to the creation of multiple objects. The client may not even be aware of this. If there is some network interruption there might not even be any response for a request. In case of the data that we operate with, the object ID is created on the client's side, which means thatthere is no need to ask for the ID generation on the server's side (in contrast to the blind following of `REST`).

Make a `PUT` request:

```
curl -X PUT -H "Content-Type: application/json" -d '{"title":"ORT"}' "http://FLUSSONIC-IP:8080/streamer/api/v3/streams/ort"
```

It is very convenient for the API that the object ID occupies only one segment. It means that if, for example, the stream name is a compound name (like `sports/football`) then you should escape the `/` character using `%2F` to access it over API:

```
curl -X PUT -H "Content-Type: application/json" -d '{"key":"value"}' "http://FLUSSONIC-IP:8080/streamer/api/v3/streams/sports%2Ffootball"
```

We call this approach `UPSERT` because it is similar to SQL `UPSERT`.

Idempotency token for POST

The idempotency token guarantees that the same operation will not be executed twice. It is especially important in a cloud platform to make sure that credits are not deducted from the account several times for the same action.

The example of idempotency token usage procedure:

1. A client uses POST request for stream creation. Each client's request includes the `Idempotency-Key` header. This is a unique value generated by the client which the server uses to recognize subsequent retries of the same request.
2. When a client creates a new stream in the cloud, the name of the stream is generated by *Flussonic* (not on the client's side) – this guarantees the stream name uniqueness.
3. So if there is some network interruption, the client has to retry the request with the same `Idempotency-Key`. *Flussonic* will recognize that it is the same request and return the stream name generated for it. On the other hand, if there is no `Idempotency-Key` in the request, *Flussonic* will consider it as a separate request and generate a new stream name.

READING OBJECTS

To **read an object** use a `GET` HTTP method:

```
curl -sS "http://FLUSSONIC-IP:8080/streamer/api/v3/streams/ort"  |jq
```

Response: JSON-encoded file with the information about the object.

```
{
  "effective": {
    "name": "ort",
    "position": 7,
    "section": "stream",
    "static": true
  },
```

```
  "name": "ort",
  "named_by": "config",
  "position": 7,
  "static": true,
  "stats": {
    "alive": false,
    "bytes_in": 0,
    "bytes_out": 0,
    "client_count": 0,
    "dvr_enabled": false,
    "dvr_only": false,
    "dvr_replication_running": false,
    "id": "61496e6e-a22f-46af-bce5-5479f9067ead",
    "input_error_rate": 0,
    "last_access_at": 1632202350648,
    "lifetime": 0,
    "opened_at": 1632202350648,
    "out_bandwidth": 0,
    "publish_enabled": false,
    "remote": false,
    "retry_count": 19,
    "running": true,
    "running_transcoder": false,
    "start_running_at": 1632202350648,
    "transcoder_overloaded": false
  }
}
```


REMOVING OBJECTS

To **delete an object**, use a `DELETE` HTTP method:

```
curl -sS -X DELETE "http://FLUSSONIC-IP:8080/streamer/api/v3/streams/ort"
```

Response: `HTTP 204` with empty body response.

Streaming API explanation

GENERAL INFORMATION

Flussonic provides a special **Streaming API** (see public reference) that allows you to build your own player or another application with all playing possibilities available in *Flussonic Media Server*.

The methods provided by this API are essentially the URLs that can be accessed by a player for playing video streams and files by various protocols (for details, see the Video Playback chapter).

Additionally to playing streams and files, with Streaming API you can:

- publish streams via some protocols;
- manage images during playback (generate thumbnails, get the logo image);
- get information about media content and DVR recording status of played streams.

AUTHORIZATION

Streaming API works in a context of a playback session with a token of a viewer. The token is inserted in the query string just as described in the Authorization chapter.

It is possible to use external authorization via the authorization backend.

EXAMPLES OF API REQUESTS

Here is the example of API request for getting HLS master playlist using the `Streaming-API: GET /{name}/index.m3u8` method:

```
curl http://FLUSSONIC-IP:8080/stream1/index.m3u8?token=60334b207baa
```

To play the stream, you should pass this link to an HLS player like STB, mobile application, or web application.

Here is the example of API request for getting a JPEG thumbnail from DVR archive using the `Streaming-API: GET /{name}/{from}-preview.jpg` method:

```
curl http://FLUSSONIC-IP:8080/stream1/1650864271-preview.jpg?token=60334b207baa
```

Streaming sessions in Flussonic

TABLE OF CONTENTS

What is a streaming session

Streaming session in *Flussonic* is a temporary and interactive information interchange between *Flussonic Media Server* and an *external system*. An *external system* refers to:

- a headend,
- a player,
- another server (either *Flussonic* or not) and etc.

Session is defined by the time interval, e.g. it is established at a certain point in time and then brought to an end at some later point. During a session at least one of the communicating parties needs to hold current state information and save information about the session history in order to be able to communicate.

A session is also defined through *type* and *states*. ***Session type*** defines where a stream is headed. From start to finish, a session goes through ***states***. A session may be initiated by the ***initiator***, however, either an initiator or a receiver may terminate it.

Here we will discuss what session types exist in *Flussonic* and what there is to know about them.

Let us consider the following example. When a viewer starts watching a TV channel, they start a new `play` session. When the viewer switches to another channel, it is considered to be a start of another session. To put it simply, ***one viewer and one channel — one session***.

Versions prior to 21.02 could track only `play` sessions via events system. If you have used the authorization system, this type of sessions should sound familiar to you.

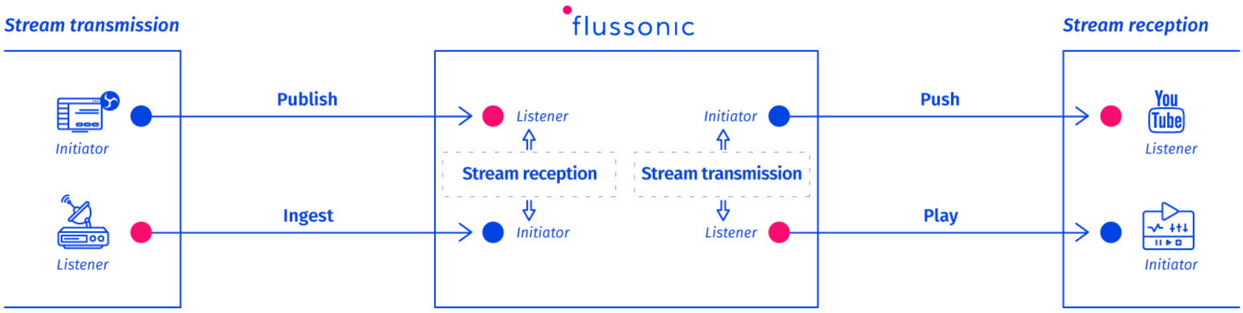
Flussonic has the following types of sessions as an addition to the previous one:

- `publish` session - when a user publishes video from webcam or OBS.
- `ingest` session - when *Flussonic* captures your source, e.g. IPTV (`udp://` , `tshttp://` , etc), IP-camera (`rtsp://`) or any other one (`rtmp://` , `shout://` , etc.)
- `push` session - when *Flussonic* pushes the stream to other server or service, like Youtube and Facebook, or performs multicast streaming.

The following table classifies these 4 types of video streaming sessions based on the initiator: `ingest` , `publish` , `play` , and `push` .

Initiated by	to Flussonic	from Flussonic
User	publish	play
Flussonic	ingest	push

Have a look at the following scheme as well:



Streaming sessions in Flussonic can be described by OpenAPI 3.0 (learn more here). You can find our public API reference with the description of all methods here.

Session lifecycle

Flussonic has a unified lifecycle for all kinds of sessions listed above. Each session has a state and transitions from one state to another, that raises an associated event.

The name of the event is made up of two elements (`session type` and `event type`), divided by an underscore (`_`).
For example: `play_opened` (session type: `play` , event type: `opened`).

For reasons of convenience `ingest` and `publish` sessions issue events under the same name: `source` .

Example

Here is an example of a `play` session:

- A user makes their first HLS request. The `play_opened` event is emitted as the new `play` session is opened.
- Authorization backend allows this session, so the `play_authorized` event is emitted.
- The player starts fetching segments, this session passes the threshold and now it is considered to be started, raising `play_started` event.
- While the user watches this stream, the `play_updated` event is emitted from time to time so that the information about this session can be saved to the Middleware.
- After some timeout since the last request the session is considered to be closed, raising the corresponding `play_closed` event.

It can be represented with the following diagram:

digraph { None -> opened [label = "play_opened", fontsize=10];

opened -> opened [label = "play_authorized", fontsize=10];

opened -> started [label = "play_started", fontsize=10]; started -> started [label = "play_updated", fontsize=10];

started -> closed [label = "play_closed", fontsize=10];

}

As we have just considered a specific instance (`play`) of session types, let's move to the general approach.

Events and session states

The diagram below represents the states that a session in *Flussonic* can possibly go through and events that are raised along this process.

> **Note**
>
> We'll name session states with a *capital letter*, while events and session types — with a *lowercase letter*.

digraph { None -> Establishing [label = "opened", fontsize=10];

Establishing -> Establishing [label = "authorized, connected", fontsize=10];

Establishing -> Finished [label = "closed", fontsize=10]; Establishing -> Running [label = "started", fontsize=10];

Running -> Running [label = "selected, updated, authorized, altered, overflowed", fontsize=10]; Running -> Finished [label = "closed", fontsize=10];

Running -> Stalling [label = "stalled", fontsize=10]; Stalling -> Running [label = " recovered", fontsize=10]; Stalling -> Finished [label = "closed", fontsize=10];

}

How do states change?

When a session starts, its state changes from `None` to `Establishing`, and the event `opened` is raised. The `Establishing` state means that the session is connecting (`connected` event), preparing and checking authorization (`authorized` event), e.g. no streaming is done yet.

A session can always end raising `closed` event and, thus, reach a state `Finished`.

Sessions `publish` and `play` can emit the `authorized` event in `Establishing` state and while `Running`.

During the `Establishing` state a session either:

- waits for the first frame or a keyframe in case it is a `source` session (`ingest` / `publish`)
- waits till there is enough bytes for the output for `play` / `push`.

Then the state is changed to `Running`, raising the `started` event.

To track the session event `updated` is emitted from time to time within the `Running` state. Use the `updated` event to update your database record for this session as it overwrites previous data about this session.

Changes in *input/output bitrate* or *media_info* in the `Running` state result in an `altered` event being emitted.

`overflowed` event can raise in `Running` state in two cases:

1. for `play` or `push`:
   If it is not possible to send the output data as quick as asked to.
2. for `source` (`ingest` / `publish`):
   If the underlying protocol informs us of that just like RTSP/RTCP or SRT do.

If the data cannot be transfered anymore, state `Running` changes to `Stalling`, raising `stalled` event. This state occurs if it is possible for the session to recover back to the `Running` state, emitting `recovered` event.

Externally initiated sessions like `play` or `publish` should pass the authorization with the help of the external authorization system. This external system must respond to periodic session pings. It can also terminate session, raising the `denied` event.

To sum up, have a look at the table below:

States transitions	Session type	Events raised
`None` -> `Establishing`	`source` (`ingest` / `publish`), `play` , `push`	`opened`
[`Establishing` , `Running`] -> `Finished`	`source` (`ingest` / `publish`), `play` , `push`	`closed`
`Establishing` -> `Establishing`	`source` (`ingest` / `publish`), `play` , `push`	`authorized` (`publish` and `play`), `connected`
`Establishing` -> `Running`	`source` (`ingest` / `publish`), `play` , `push`	`started`
`Running` -> `Running`	`source` (`ingest` / `publish`), `play` , `push`	`selected` , `updated` , `authorized` , `altered` , `overflowed`
`Running` -> `Stalling`	`source` (`ingest` / `publish`), `play` , `push`	`stalled`
`Stalling` -> `Running`	`source` (`ingest` / `publish`), `play` , `push`	`recovered`

How to configure events

Add the `event_sink` section to the configuration file (`/etc/flussonic/flussonic.conf`).
Here is an example:

```
event_sink example {
  url http://examplehost:5000/events;
  only media=example_stream;
}
stream example_stream {
  input fake://fake;
}
```

With this configuration an HTTP POST requests with JSON body will be sent, including sessions described in the section above.

For more information on configuring events handlers, see Configuring event logging.

Source connection events

In the example below you can see a series of events, when *Flussonic* connects to the source:

- `source_connected` — an HTTP connection (`"status":"http_connect"`) started.
- `source_started` — `source_id=7ad153b1-68a5-4304-bbfd-b136603baebd` was created.
- `stream_updated` — `bytes` , `bytes_out` for the `source_id=7ad153b1-68a5-4304-bbfd-b136603baebd` were updated.

```
[{
  "event":"source_connected",
  "event_id":1023,
  "id":"4f3c7cec-5c36-4670-921b-a0dcd4a6f0c8",
  "loglevel":"info",
  "media":"example",
  "priority":1,
  "proto":"tshttp",
  "server":"mk1.e",
  "status":{"status":"http_connect"},
  "url":"tshttp://127.0.0.1/fake/mpegts",
  "utc_ms":1614524093408
 },{
  "dts":93606612.44444445,
  "event":"source_started",
  "event_id":1027,
  "id":"4f3c7cec-5c36-4670-921b-a0dcd4a6f0c8",
  "loglevel":"info",
  "media":"example",
  "priority":1,
  "proto":"tshttp",
  "server":"mk1.e",
  "source_id":"7ad153b1-68a5-4304-bbfd-b136603baebd",
```

```
  "url":"tshttp://127.0.0.1/fake/mpegts",
  "utc_ms":1614524093414
 },{
  "bytes":0,
  "bytes_out":0,
  "event":"stream_updated",
  "event_id":1028,
  "id":"82c59180-e64e-42fc-8f11-2dec111ca5f7",
  "loglevel":"debug",
  "media":"example",
  "opened_at":1614524093399,
  "server":"mk1.e",
  "source_id":"4f3c7cec-5c36-4670-921b-a0dcd4a6f0c8",
  "utc_ms":1614524093415
  }]
```


Playback started event

The event `play_opened` is raised when a client connects to an HLS stream:


```
[{
  "bytes":0,
  "country":null,
  "event":"play_opened",
  "event_id":1064,
  "id":"24b79b95-7400-4da2-bf9c-a855603baed1",
  "ip":"192.168.100.7",
  "loglevel":"debug",
  "media":"example",
  "opened_at":1614524113129,
  "proto":"hls",
  "query_string":"token=test",
  "referer":null,
  "server":"mk1.e",
  "source_id":"82c59180-e64e-42fc-8f11-2dec111ca5f7",
  "token":"test","user_agent":"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.192 Safari/537.36",
  "user_id":null,
  "user_name":"example",
  "utc_ms":1614524113129
 }]
```


Note that `"source_id":"82c59180-e64e-42fc-8f11-2dec111ca5f7"` is the same ID as in the previous example. All events are connected with each other through the `source_id` parameter.

Events associated with sessions are listed in the API schema.

Manage events with API

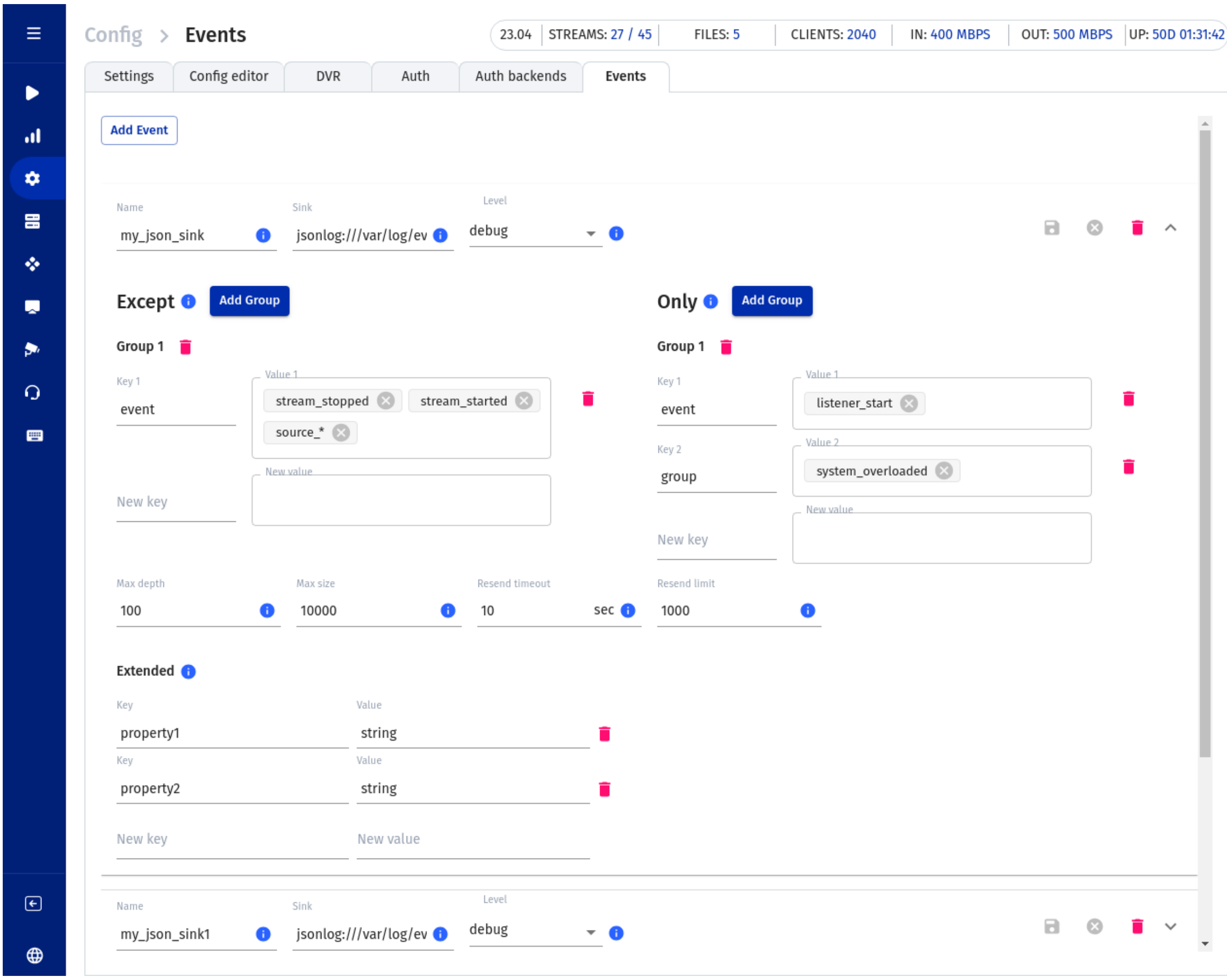
EVENTS IN FLUSSONIC

Flussonic has a system of internal events with routing and handling, and convenient and flexible tools to configure it. This page describes how to configure *Flussonic* to filter and send events. You can find the list and descriptions of all the events in the API schema.

It would also be useful for you to read more about streaming sessions.

Events are initiated in different parts of the system and can be used in different scenarios.

To configure event-related settings, go to **Config** -> **Events** or add the `event_sink` directive into the *Flussonic* configuration file.



In the **Sink** (or `url` in the config file) option, define the receiver of events:

- To use your custom handler, specify the path to the handler in `url` .
- To write events to a log file, specify the path to the file in `url` .

Then use various options to filter events before they come to a handler or log.

Table of contents:

CONFIGURING EVENT LOGGING

In addition to the main log, *Flussonic* allows you to create as many log files as you need and to log events according to your filtering settings. This feature can be configured at **Config** -> **Events**.

The same settings can be performed in the config file by adding the `event_sink` directive and specifying the file in `url log://` option, for example:

```
event_sink log_name {
    url log:///var/log/flussonic/crash.log;
    level debug;
}
```

The main settings are:

- **Name** (`log_name`) is just the setting's name. It's good to give it a meaningful name.
- **Sink** (`url`) is the file where event information is logged. See also Configuring event handlers.
- **Level** (`level`) is the level of logging according to event importance. Can be `debug` (the most detailed logging), `info` , `alert` (only serious events), `notice` , `warning` , `error` , `critical` .

CONFIGURING EVENT HANDLERS

Flussonic can send events to any recipient you specify in the **Sink** (`url`) parameter. This may be the path to the script file, the URL of the remote handler, and so on. For example:

```
event_sink handler_name {
    url http://IP-ADDRESS:PORT/SCRIPT_NAME.php;
}
```

Such configuration creates an event handler with the name `handler_name` and it sends all the events to HTTP URL `http://IP-ADDRESS:PORT/SCRIPT_NAME.php` .

In this configuration all *Flussonic* events will be send in JSON format as a list of objects. On a high loaded system it can generate enormous amount of events most of which are not required. Try using filtering to reduce the event traffic.

Event handler calls are synchronous: an event will not be sent to the handler if the handler hasn't handled the previous event batch.

The event configuration block supports the following configuration options:

Option	Description
`url`	The specification of the handler. It can be `http://URL` , `https://URL`
`only`	The white list of limitations. You can specify several `key=value` or `key=value1,value2` options on each `only` line. You can filter events by their `event` field, by `media` field or any other like `country` or `ip` . Usually it is `event` and `media` . You should read more explicit explanation of this `only` behaviour.
`except`	The black list of limitations. Events matched by any of `except` fields will not be passed to handler.
`buffer`	Not recommended.
`sign_key`	(Set in the **Extended** (`extra`) section) You can specify signature key for HTTP event sink. When *Flussonic* prepares HTTP POST with JSON body, it will add this secret key to the end of the body, making SHA1 hash from it and adding it in hex form as a header X-Signature. This can be used for verifying that it is *Flussonic* posting events.

All other configuration options in this block will be passed to the specified sink handler. See the full list of options in the description of the API call: `Flussonic-API: PUT /streamer/api/v3/event_sinks/{name}/` . In a LUA script they can be accessed via the `args` table. When using HTTP backend you pass them along with other parameters.

EVENT FILTERING

You can pre-filter events before passing them to handlers or files with **Except** (`except`) and **Only** (`only`) options. This mechanism reduces the load on your event handler. Each event is prefiltered in the emitter thread before being passed to the handler.

Rules for filtering:

- if any `except` directive fully matches event, it is dropped and not sent to handler;
- if there are no `only` directives, events are sent to handler;
- if there are `only` directive then event is passed to handler if ANY directive fully matches the event.

Full match of an event and a directive means that all `key=value` pairs in directive are equal to values in event. If a directive has a `key=value1,value2,value3` pair, then it means that the event must have any of these values to match this directive.

Examples:

- `only event=stream_opened;` matches `{event: "stream_opened", media: "cbc"}`
- `only event=stream_opened,stream_closed;` matches `{event: "stream_opened", media: "cbc"}`
- `only event=stream_opened,stream_closed media=tnt;` **does not** match `{event: "stream_opened", media: "cbc"}`
- `only event=stream_opened media=cbc group=news;` **does not** match `{event: "stream_opened", media: "cbc"}`

For example, the following configuration will not write to the log all events concerning streams (and write other events, such as *Flussonic* server events):

```
event_sink log_name {
    url log:///var/log/flussonic/events.log;
    except media=*;
    level debug;
}
```

Example of configuration to send only certain events to the handler:

```
event_sink handler_name {
    url http://IP-ADDRESS:PORT/SCRIPT_NAME.php;
    only event=stream_opened,stream_closed,source_opened,source_closed;
}
```

RELIABLE DELIVERY OF EVENT NOTIFICATIONS

To prevent notifications loss, you can set up *Flussonic* for postponed attempts to resend notifications. If the receiving HTTP server or script does not respond, *Flussonic* accumulates events in a special buffer and periodically retries sending them. When the receiving server responds, *Flussonic* will send all the accumulated notifications.

For this, specify two options:

- **Resend limit** `resend_limit` is the number of the most recent events that will be stored in order to retry sending them. Cannot exceed 2000.
- **Resend timeout** `resend_timeout` is the time interval, in seconds, over which *Flussonic* will try to send events again.

In the config file, it will look like this:

```
event_sink watcher {
   url http://IP-ADDRESS:PORT/SCRIPT_NAME.php;
   resend_limit 10;
   resend_timeout 10;
}
```

ADVERTISEMENT LOGGING

You can use the `ad_injected` event to log information about injecting and playing advertisement in live streams. Each time advertisement is inserted into a played stream, *Flussonic* generates this event with such parameters as the first advertisement frame DTS, the path to the advertisement files, advertisement type and duration. For details please refer to the API schema (select `ad_injected` in the `events` parameter of the response).

This event is written to *Flussonic* log allowing to monitor, how much advertisement have *Flussonic* showed and to whom.

3. Media Server

3.1 General

3.1.1 Glossary

Here you can learn about terms and concepts that you meet in the documentation on Flussonic Media Server.

Adaptive bitrate streaming

Adaptive bitrate is a method of video streaming over HTTP where the source content is encoded at multiple bitrates.

Deinterlacing

Deinterlaicing is converting an interlaced video to a progressive video.

The interlaced video demonstrates even and odd scan lines as two individual fields. At first, the even lines pass on the screen and then the odd lines pass. Two of such even and odd scan line fields make one video frame. Interlaced videos are great for broadcasting as video images can be processed onto the screen with very little bandwidth. The drawback of interlaced video is that in fast motion, it may be blurred as only half of the image is captured at a time, movement along the frame causes motion artifacts.

Progressive video content shows the even and odd scan lines, that is the entire video frame on the screen at the same time.

Deinterlaicing is necessary for comfortable viewing of legacy TV video on PC/mobile devices.

DVR

This is a set of Flussonic features related to recording streams to an archive and then playing the archive via different protocols or export the selected part of it to an MP4 file.

Frame

For video, a frame is one of the many still images which compose the complete moving picture. It is the minimal piece of a video track. Each frame has a start time and a duration.

Frame duration

For video track, it is the time between the beginning of a frame and the beginning of the next frame.

This parameter is important for some protocols. Normally, frame duration is a difference between timestamps of two neighbouring frames. However, sometimes (when the connection is broken) video breakups are possible. As a result, the delta between two consequent frame timestamps will not be equal to the frame duration. This situation is considered as a frame gap and is handled differently across different protocols.

GOP (Group of Pictures)

Group of Pictures (GOP) — a structured group of successive frames in an MPEG-encoded video stream. Frames are grouped for the interframe compression purposes. We need compression to transfer video over networks. The encoder software compresses video data to reduce its amount compared with non-compressed (raw) video data.

A compressed stream is a succession of GOPs. On a receiving side, the decoder takes all frames in a GOP and creates an image that you can see.

A GOP consists of an I-frame followed by P-frames and B-frames:

- I-frame (keyframe) – is a first frame in a GOP. It is a full image encoded independently from other frames (meaning no links to them). Each GOP has a keyframe at the start.
- P-frame, B-frame – frames that go after the keyframe in a GOP.
    - P-frames contain the difference between the previous P-frame and a current frame. It is encoded with a link to an I-frame.
    - B-frames contain links to I-frames and P-frames before and after themselves. It helps to rewind quicker, for example.

GOP size

GOP size (the number of frames between two neighbor keyframes) – the number of frames in one GOP. This number can be variable or constant for a stream. When Flussonic transcodes a stream, it creates GOPs of a constant size, so all GOPs have the same size.

Multicast

Multicast is a method of video distribution in a local network. A multicast is a set of UDP packets transmitted from the same source to a group of subscribers at one time. A special multicast IP address is used.

Learn more in the Flussonic documentation

Prepush

Prepush is a method used to achieve a smoother playback of HTTP MPEG-TS, RTMP, or RTSP video streams transmitted via TCP.

With prepush, a streaming server saves each GOP in the buffer before sending it to a client. When a client connects to the server, the server sends the first GOP from the buffer and then transmits a stream with a timeshift — the delivery lags behind for a time interval equal to the size of one GOP converted to seconds. When the connection with the server breaks or slows down, the client plays a GOP from the buffer. In this way, video is played more evenly.

Publishing

Publishing is transmitting video to Flussonic Media Server from external systems and devices that initiate the connection. Flussonic is the party that awaits the connection.

What we call publishing to Flussonic:

- Transmitting video from a mobile device to Flussonic.
- Transmitting video from OBS (Open Broadcaster Software) or vMix to Flussonic. Learn more
- Transmitting video from a webpage to Flussonic via WebRTC. Learn more

And this is what we don't call publishing:

- Receiving a multicast
- Ingesting a stream from some source (in this case it's Flussonic that initiates the connection).

Segments

DASH and HLS protocols break a stream into segments, or chunks. These chunks have a constant duration measured in seconds. Segments are used for transfer and buffering purposes. A segment can contain several GOPs and it must be divisibale by GOP. A segment cannot be shorter than a GOP.

The sender of a DASH or HLS stream transfers video segment by segment, and it sends to a client a so called playlist that is the list of segments. Before the start of playing a stream, the client saves some segments in the buffer. If the connection with the server breaks or slows down, the client plays video segments from the buffer, so video is played more smoothly. The client usually downloads three segments before it strats the playback.

Transcoder

Transcoder is a component that performs the direct digital-to-digital conversion of the initial video stream to provide a multi-bitrate stream, to change video parameters (codec, image size, bitrate), or to place a logo on top of a video stream.

Video codec

It is a technology for compressing raw video for subsequent packaging into a container, which, in turn, will be used for delivery via a specific streaming protocol.

Video container

A container (or transport) is the format for packaging encoded data in a file or a stream for transmission over a network. Packets with audio and video data are transmitted at the transport layer according to the OSI model.

The container format is self-sufficient and independent of the delivery protocol, that is, you can package the data and play them on your local machine, and not necessarily transfer them over the network.

Video streaming protocol

Video streaming protocol is the rules for the exchange of data, commands and responses to them between two participants in a video communication (client-server or peer-to-peer).

When preparing data for transmission over the network:

- First, the video and audio data must be compressed
- Then they must be packed in a container for streaming via a certain protocol.

3.1.2 Flussonic data model

What is media in Flussonic

Media is a stream or a file played in *Flussonic*. Each media has a name and a set of data.

To manage media data conveniently and effectively, we use a data model that allows to divide media into separate elements. This data model is the same for streams and files. Let us consider the parts of media in *Flussonic*.

Parts of played media

TRACK

Each media can be divided into separate **tracks** that represent video, audio, or text (e.g., subtitles). For example, a movie can contain one video track, three audio tracks (English, German, and Russian), and three corresponding subtitle tracks.

Each track is characterized by its **content**, i.e., physical substance (video, audio, or text), and several other parameters. The set of track parameters depends on its content. For example, a video track can have width and height of the displayed image as well as **frame rate** — the speed at which a sequence of images is displayed on a screen. Audio track can have other parameters, such as language and **sample rate** — the number of samples per second taken from a continuous signal from a microphone (or another audio source) to make a discrete or digital signal. Text tracks are very simple and don’t have any specific parameters.

Flussonic automatically assigns an identifier to each track, for example, "v1, v2, …" — for video tracks, "a1, a2, …" — for audio tracks, "t1, t2, …" — for WebVTT subtitle tracks, "l1, l2, …" — for DVB subtitles or teletext.

Each track, independent on its content, can be divided into **frames**.

FRAME

Frame is the minimal piece of a track. A frame can be a part of video, audio, or text track. For a video track, a frame is one of the many still images which compose the complete moving picture. Each frame has a start time and **frame duration**. Frame duration has a different meaning for audio and video.

For an audio track, frame duration depends on **sample rate**. For example, CDs are usually recorded at 44.1 kHz – which means that every second 44,100 samples are taken. In this case 1/44100 seconds can be considered as an audio frame duration.

For a video track, frame duration is the time between the beginning of a frame and the beginning of the next frame. This parameter is important for some protocols. Normally, frame duration is a difference between timestamps (start times) of two adjacent frames. However, sometimes (when the connection is broken) video breakups are possible. As a result, the delta between two consequent frame timestamps will not be equal to the frame duration. This situation is considered as a frame gap and is handled differently across different protocols. For example, HLS protocol will continue playback, however DASH protocol will break the playback and start a new period (learn more here).

The important feature of *Flussonic* data model is that frames never overlay each other. Overlaying frames can result, for example, in such a problem as subtitle overlapping. *Flussonic* allows to avoid such a problem because a frame cannot start earlier than the start of the previous frame + its duration.

GOP

To deliver video over the internet using a limited bandwidth, it is often necessary to compress the video. Besides compressing frames themselves, there is a more progressive technology called interframe compression. It works by sending full frames (referred to as **keyframes**), and then only sending the difference between the keyframe and the subsequent frames. The receiver (decoder) uses the keyframe plus these differences to re-create the desired frame with reasonable accuracy.

For interframe compression purposes, frames in a track are grouped into **GOPs**. GOP (group of pictures) is a structured group of successive frames in a video stream or file. Each GOP consists of an I-frame (keyframe) followed by P-frames and B-frames:

- **I-frame (keyframe)** is the first frame in a GOP. It is a full image encoded independently from other frames (meaning no links to them). Each GOP has a keyframe at the start.
- **P-frames** contain the difference between the previous P-frame and a current frame. It is encoded with a link to an I-frame.
- **B-frames** contain links to I-frames and P-frames before and after themselves.

A typical GOP contains a repeating pattern of B- and P-frames following the keyframe. An example of a typical pattern might be the following:

I B B P B B P B B P B B

Ideally, keyframes should be selected when a scene changes (so called *scene detection* method). However, most programs for processing video are configured to work with GOPs of equal size. Therefore, in most situations equal GOPs are used, for example, the TV standard is 28 frames in a GOP.

It is important to understand that a GOP without a keyframe has no sense. Thus, it is impossible to play video just in the middle of the GOP.

Grouping into GOPs is applicable to video frames only. Corresponding audio and text subtitle frames are added to GOPs synchronously.

What would be the optimal GOP length?

Why a GOP should not be too long? Because a longer GOP can result in a bigger *zap time* – duration of time from which the viewer changes the channel using a remote control to the point that the picture of the new channel is displayed. If a viewer clicks the remote control before the previous GOP has finished, they see unactual picture. This problem may be critical for video games or video calls.

To solve this problem, *Flussonic* uses the **prepush** feature: it saves each GOP in the buffer before sending it to a client. When a client connects to the server, the server sends the first GOP from the buffer and then transmits a stream with a timeshift — the delivery lags for a time interval equal to the size of one GOP converted to seconds. When the connection with the server breaks or slows down, the client plays a GOP from the buffer. In this way, video is played more evenly, however, a latency may grow.

Why a GOP should not be too short? Because longer GOPs provide better compression.

Different applications use different GOP lengths, but typically these lengths are in the 0.5 – 2 second range.

Open GOP

In some cases, it is possible to compress video even better by using so called **open GOPs**. Open GOP contains P-frames that refer to the frames before the keyframe. This allows to lower bitrate by 5-7 %. However, open GOP may result in problems when it comes to using **segments**.

SEGMENT

Segment is the next-level element in our data model. It contains one or more GOPs with corresponding audio and text frames (synchronized with video frames). Segments are necessary for some protocols, such as HLS and DASH.

Sometimes a segment can match a GOP, sometimes not. However, it is always divisible by a GOP and starts with a keyframe.

The important feature of the transcoder in *Flussonic* is that all segments for all video tracks are always synchronized. When encoding video with some other (not very good) transcoder, it is possible that one video track has already started playing a new segment, while another video track is still playing the previous segment. In this case it may be difficult for a player to switch to another video track, therefore such situation is unacceptable for multibitrate streaming. In *Flussonic*, all segments have the same size and the same timestamps as the corresponding segments in another video track. That is why all video tracks are played synchronously.

Please note that audio frames have another frame duration than video frames, so it is possible that when a new segment starts playing, audio frames are still being added to the previous segment. This is a normal situation.

We store segments isolated from each other. Sometimes it may result in problems when using open GOPs because a P-frame cannot refer to frames from the previous segment. In this case a picture may sometimes break up, so it is necessary to wait for the next segment for better picture quality.

3.2 Ingest

3.2.1 Requirements for source streams and files

This page describes the requirements and recommendations to input streams and files. Below you will see a list of streaming protocols, media containers and codecs supported by *Flussonic Media Server* for input streams from sources.

Features of input streams and sources

Stream format is defined by the source, but sometimes the format can be selected in the source settings, like for a hardware encoder or a video camera. When the stream format cannot be selected, transcoding is required (see more below).

Flussonic supports a limited set of codecs and containers. If an input stream has unsupported codecs or containers, *Flussonic* throws an error. Playing back such streams can cause issues, for example, with DVR playback.

Flussonic distinguishes between stream (live) and file (VOD) formats.

Input stream formats

The table below shows a list of streaming protocols and their corresponding containers, video, and audio codecs supported by *Flussonic Media Server* for incoming streams. Find the list of currently supported codecs for the input streams in the Flussonic Media Server API Reference.

Streaming protocol	Media container	Video codec	Audio codec
HLS	MPEG-TS	H.264, H.265 (HEVC), AV1	AAC, EAC3, MP3, AC-3
WebRTC	-	H.264, VP8, VP9, AV1	Opus, PCMA (G.711 A-law), PCMU (G.711 μ-law)
RTMP	FLV	H.264	AAC, MP3, PCMU (G.711 μ-law)
RTSP	-	H.264, H.265 (HEVC)	AAC, PCMA (G.711 A-law), PCMU (G.711 μ-law)
RTP	-	H.264, H.265 (HEVC)	-
HTTP/UDP	MPEG-TS	H.264, H.265 (HEVC), MPEG-2 video	AAC, EAC3, MP3, MPEG-2 audio
SRT	container-agnostic	H.264, H.265 (HEVC), MPEG-2 video, VP8, VP9, AV1	AAC, EAC3, MP3, AC-3, Opus, MPEG-2 audio, PCMU (G.711 μ-law), PCMA (G.711 A-law)
Shoutcast/ ICEcast	-	-	AAC, MP3
H323	-	H.264, H.265 (HEVC)	PCMA (G.711 A-law), PCMU (G.711 μ-law)

Read more about publishing video from a browser to *Flussonic* via WebRTC in the WebRTC Publishing.

Source file formats

The table below shows containers, video, and audio codecs supported by *Flussonic Media Server* for source files.

Media container	Video codec	Audio codec
MP4 (.mp4, .f4v, .mov, .m4v, .mp4a, .3gp, .3g2)	H.264, H.265 (HEVC)	MP3, AAC (all profiles)

Learn about VOD files in the VOD Files.

Frame rate per second (FPS)

If an incoming stream has a frame rate less than 10 FPS, *Flussonic* considers the source inactive. We recommend to stream at 15 FPS or higher.

Some CCTV security cameras output streams in fragments of a couple of seconds long because of the low frame rate settings of the camera. In this case, you need to increase the FPS in the camera settings.

> **Warning**
>
> Growing FPS affects the security camera performance. So some cameras may overheat, freeze, being unable to output a stable stream. It is crucial to balance the video quality and performance for such cameras. One way to do this is to reduce the number of simultaneous connections to the camera to 1 so that only *Flussonic Media Server* receives the stream.

How to get the necessary format

Sometimes the format of the incoming stream cannot be selected in the source settings (like for a stream from a satellite). In this case, it is necessary to use transcoding.
Flussonic has a built-in transcoder, which can convert streams from various formats to H.264/AAC and more.

To learn more about transcoding and how to configure it in *Flussonic*, see Transcoding and Transcoder.

Thus, *Flussonic* can receive an input stream without transcoding, if:

1. The stream format is supported.
2. Stream is tt least at 10 FPS or preferably 15 (and higher) FPS.

If your stream does not meet the above requirements, you can use the built-in transcoder to convert the input stream to the necessary format.

3.2.2 Source failover

For various reasons, a video source may temporarily disappear or even become offline. To avoid a situation where consumers do not have any video, it is necessary to prepare alternative stream sources in order to broadcast them in the absence of the main source until it is restored. *Flussonic* provides seamless automatic source switching.

On this page:

Redundant sources

To maximize service uptime for your subscribers, you can use the Source Failover feature. By specifying multiple sources, you instruct *Flussonic* to automatically failover to the secondary data sources if the primary source becomes unavailable. *Flussonic* supports using video streams and files as secondary sources.

Source switching happens when the stream becomes disconnected, or when there are no incoming frames from the source for more than 60 seconds (and 180 seconds for `hls://`, `playlist://`, `timeshift://` sources).

How source failover works

After *Flussonic Media Server* switches to a secondary source, it will periodically check if the first source is up. When the first source comes back online, *Flussonic* will fall back to it.

> **Note**
>
> Flussonic waits for a keyframe from a reappeared source and only then switches to that source. In this way, we provide seamless switching without delays. This is essential for video with a large GOP, for example, video via HLS — Flussonic provides high-quality switching even for such video.

> **Warning**
>
> Secondary sources MUST have the same set of audio and video tracks as your primary source if you want to achieve the most stable output and best user playback experience.

Example

The stream `example_stream` has two sources. If no frames come from the first source for 20 seconds, then *Flussonic* will switch to the second source.

```
stream failover_example_stream1 {
  input udp://239.0.0.1:1236 source_timeout=20;
  input tshttp://localhost:80/clock2/mpegts;
}
stream clock1 {
  input fake://fake;
  push udp://239.0.0.1:1236 multicast_loop;
}
```

```
stream clock2 {
  input fake://fake;
}
```

Failover conditions

Flussonic monitors only the time since last frame was received from the source, and switches to another source if there were no incoming frames received within a certain timeframe.

Flussonic doesn't monitor conditions like video or audio loss or increased volume of MPEG-TS CC errors.

Options for configuring source failover

source_timeout

The `source_timeout` option specifies the period of time, in seconds, for which *Flussonic* will wait for new frames until it considers the source as lost. The default timeout is 60 seconds (180 seconds for hls://, playlist://, timeshift:// sources).

You can specify `source_timeout` for both the entire stream and for each of the video sources. The `source_timeout` option of a video source has priority over the `source_timeout` option of its parent stream. Example:

```
stream backup_timeout {
    input publish:// source_timeout=10;
    input fake://fake source_timeout=5;
    source_timeout 20;
}
```

If you think that switching occurs too often, you can increase the `source_timeout` so that there are no "jumps" from one source to another. On the other hand, in order not to wait for a long time until *Flussonic* switches to another source, you can reduce the timeout.

The timeout is not taken into account when you switch sources manually.

priority

You can assign priorities to stream sources, and *Flussonic* will take priorities into account when switching to another source. The source with `priority=1` has the first priority, the source with `priority=2` has the second priority, and so on.

By default, the first source in the list has the highest priority and the last source in the list has the lowest priority. If priority is not specified for some sources, then the default order is applied.

Flussonic checks priorities only after it determines all sources that are active.

If the priority of an unavailable (offline) source is equal to the priority of the currently played source, then *Flussonic* will not try to fall back to the source that has become unavailable.

```
stream example_stream {
  input fake://fake priority=2 source_timeout=30;
  input tshttp://10.2.4.5:9000/channel/5 priority=1 source_timeout=10;
}
```

The rules of switching sources according to their priority and state (whether a source is available or not) apply to published sources as well as any other ones.

Changing sources manually

Flussonic supports manual source switching.

To change the source of a stream manually, without waiting for the timeout:

- In the stream settings, change the order of sources. Use this if `priority` was not specified.

For example, move the second source up, and *Flussonic* will switch to it.

- In the stream settings, edit the priority of sources

For example, set priority=2 instead of priority=1 and priority=2 instead of priority=1, and *Flussonic* will switch to the source with the highest priority.

- Enable another source via the API.

Recording to an archive

If a DVR location is configured, *Flussonic* will start archiving video from the active source.

The system makes no distinction between live sources and local video files. If *Flussonic* has switched to a file source, the contents of this file will be written to archive.

It is possible to use a static video as a failover data source.

```
stream backup_example_stream1 {
  input tshttp://example.com/origin;
  input file://vod/bunny.mp4;
  dvr /storage;
}
```

In the example above, the fallback video file would be written to the DVR archive. To avoid writing a fallback video file to the archive when all of the sources are down, you should use the backup directive instead of a static video URL. See also further on this page.

Using a file as a redundant source

You can use static video (video files) as a failover data source.

Files can be specified in two different ways, each leading to a certain behavior of *Flussonic* at source failover.

Using the `input file://` schema to list a file as *one of stream sources*

```
  stream backup_example_stream2 {
    input tshttp://10.0.4.5:9000/channel/5;
    input file://vod/bunny.mp4;
  }
```

Flussonic supports MP4 and MPEG-TS files (.ts).

Using the `backup` option to set a file as the *failover data source*

To set a file as a failover data source for the main stream, use the `backup` option. *Flussonic* shows this file without actually switching sources. This is useful in certain cases. **Learn more**

```
  stream backup_example_stream3 {
    input tshttp://10.0.4.5:9000/channel/5;
    backup vod/bunny.mp4;
  }
```

How 'backup' is different from 'input file://'

Unlike source switching with the source `input file://<VOD location>`, when a fallback file is used, *Flussonic* technically does not switch to another source. This is especially useful for published streams to prevent numerous closings of a socket with a publishing client.

The fallback file specified in `backup <VOD location>` is not transcoded and not written to DVR, unless you configure otherwise. The file source `input file://<VOD location>` is always written to DVR.

When use `backup` instead of `input file://`:

- In case of poor connection with the client that publishes video, *Flussonic* continues to receive frames without interrupting the connection with the client. This allows the client to continue the publishing session without having to start it over each time the source was switched. When the published stream disappears, viewers see the fallback file and understand that the broadcast is not over yet.
- When all sources are unstable and *Flussonic* switches between them too often, it is better to show a fallback file. If you use a file as one of the sources, viewers will see any video only after timeouts pass for each of the troubled sources.
- If you write the main stream to DVR and do not want to write the file too in order to prevent the file from appearing in the archived video.
- Using options like timeout for the main stream and the fallback file, you can manage which source to show during a publication session.

Failover file options (`backup`)

The fallback file takes the following options:

```
stream example {
    input udp://239.0.0.1:1234;
    backup vod/bunny.mp4 video_timeout=5 audio_timeout=10 timeout=20 dvr=true transcode=true;
    dvr /storage;
}
```

dvr=true

If the main stream has a configured DVR, then the fallback stream will be recorded to the archive too:

```
stream example {
    input udp://239.0.0.1:1234;
    backup vod/bunny.mp4 dvr=true;
    dvr /storage;
}
```

timeout=10

The time (in seconds) for *Flussonic* to switch to the fallback source if the main source stops sending frames. The important thing here is that the source remains active (connected), allowing for a client-publisher to stay on the socket.

This option takes any type of frame into account.

Flussonic can switch to a fallback source only when there are no frames of a certain type (video or audio) coming, which allows better control of source switching. You can different timeout intervals for different frame kinds. To take into account only audio or only video frames, use the options `audio_timeout` or `video_timeout` (see further in this list).

If you do not specify `timeout` specifically for a fallback source, then in the absence of frames, `source_timeout` of the main source will be used.

By using `timeout` and `source_timeout` together, you can:

- Set a longer timeout so that mobile clients manage to start streaming without being disconnected
- At the same time, switch to the fallback source as soon as possible.

For example, a WebRTC client app will stay connected for the specified time when the source stops sending frames. The fallback file is played during this time.

Example:

```
stream example {
    input publish:// source_timeout=20;
    input fake://fake;
    backup vod/bunny.mp4 timeout=1;
}
```

In this example:

Before the publication begins, the `fake` stream is played. Then the client app connects to *Flussonic* to stream video to it. If, after the connection was established, no frames arrived from the client during 20 seconds, then the client is forcibly disconnected and the demo source starts playing.

After the start of publication, if for 1 second there are no frames from the published stream, the file `backup-file.mp4` starts to play. However, the publication source is not disconnected yet.

When the source resumes sending frames, the stream switches to the publisher client and the published video is played. However, if the source does not resume sending frames during 20 seconds, then the publisher client is forcibly disconnected and the demo source starts playing.

video_timeout=5

The time (in seconds) for *Flussonic* to switch to the fallback source if the main source stops sending video frames.

If you specify `video_timeout`, `audio_timeout` and at the same time `timeout` of the main source, switching will be triggered by a timeout, which will occur first. These options have the same priority.

audio_timeout=10

The time (in seconds) for *Flussonic* to switch to the fallback source if the main source stops sending audio frames.

If you specify `video_timeout`, `audio_timeout` and at the same time `timeout` of the main source, switching will be triggered by the timeout that occurs first. These options have the same priority.

transcode=true

See Transcoding the Failover File further on this page.

Transcoding the failover file

transcode=true

If the main stream is transcoded, then the fallback file will be transcoded too with the same parameters as the main stream.

```
stream backup_transcode {
    input udp://239.0.0.1:1235;
    source_timeout 5;
    backup vod/bunny.mp4 transcode=true;
    transcoder vb=1000k ab=64k;
}
```

This allows you to change transcoding parameters without the need to transcode the fallback file with new parameters. This also makes it unnecessary to prepare several fallback files with different bitrates.

Emergency button

You can configure an emergency button in *Flussonic*.

***Emergency button** is a mechanism, used to manage the start/stop of the stream source.*

Flussonic checks emergency button status before starting the stream source.

To set an emergency button you should specify a path to the file, that will be providing information, using one of the two parameters for the source `url` in the stream settings: `allow_if` or `deny_if`.

Under what circumstances will the source start?

- File specified through **`allow_if` contains 1** in it (`allow_if=/PATH/TO/FILE` carries 1).
- File specified through **`deny_if` contains 0** in it (`deny_if=/PATH/TO/FILE` carries 0).

Therefore, in all the other cases the source will not start. Such as:

- File specified through `allow_if` **contains 0** in it (`allow_if=/PATH/TO/FILE` carries 0).
- File specified through `deny_if` **contains 1** in it (`deny_if=/PATH/TO/FILE` carries 1).
- File contains any other values except from 1 and 0.
- File does not exist.

The confuguration may look as follows:

```
stream example_stream {
    input m4f://FLUSSONIC-IP/STREAM_NAME deny_if=/PATH/TO/FILE;
    input udp://239.0.0.2:1236 allow_if=/PATH/TO/FILE;
}
```

In the example above we set 1 emergency button (`/PATH/TO/FILE`) to 2 sources (`m4f://` and `udp://239.0.0.2:1236`). If `/PATH/TO/FILE` contains 1, then `m4f://` source will not start, whereas `udp://239.0.0.2:1236` will. Hence, if `/PATH/TO/FILE` contains 0, then `m4f://` will start, whereas `udp://239.0.0.2:1236` will not.

Summing up ("+" — source starts, "-" — source stops):

file status	allow_if	deny_if
File contains 1	+	-
File contains 0	-	+
File contains another values	-	-
File does not exist	-	-

3.2.3 Live streaming

Flussonic Media Server can retransmit streaming video into multiple output formats on the fly with just-in-time packaging. For example, you can ingest an MPEG-TS stream, deliver it simultaneously to thousands of subscribers in DASH or HLS format, and at the same time publish the stream via RTMP on YouTube.

Flussonic Media Server supports three types of streams:

- `static` — streams that are being broadcasted all the time.
- `ondemand` — streams that are requested by users (on-demand).
- `live` — user-published streams. See Publishing for details.

Contents:

Static streams

Static streams are launched upon a start of the server. Flussonic continuously monitors static streams.

If for some reason (transcoder went off, antenna broke down) a data source goes down, Flussonic Media Server will constantly keep trying to reconnect to the stream until success or shutdown.

Usually, IPTV channels or IP camera feeds are being configured as a static stream.

Flussonic Media Server supports many types of data sources, which must be configured with URLs.

The format of a stream definition in the **/etc/flussonic/flussonic.conf** file is:

```
stream example_stream {
  input udp://239.0.0.1:1234;
}
```

In this example:

- `example` is the name that must be used to request the stream from Flussonic Media Server.
- `udp://239.0.0.1:1234` is the data source URL.

Important. The name of a stream should contain only Latin characters, digits, dots (`.`), dashes (`-`), and underscores (`_`). If the name contains any other characters, DVR and live streams might work incorrectly.

To add a stream via the web interface:

Go to the **Media** tab and click **Add** next to **Streams**.

Then enter the name of the stream and the data source URL. Click **Create** and Flussonic will add the stream to the list.

Note. By default, new streams are **Static**. To change the stream type to **On demand**, click **Static** next to the stream name.

After the stream is added, you can go to the stream settings page and check the ingest status:

On-demand streams

If the stream is not needed all the time but only upon user's request, you can configure Flussonic Media Server to turn it off when it is not being used and turn it back on when it is requested.

To specify this kind of behavior, change the stream type to `ondemand`:

```
ondemand ipcam {
  input rtsp://localhost:554/source;
}
```

> **Warning**
>
> If Media Server ingests the `ondemand` source stream using RTMP, RTSP, or HTTP MPEG-TS protocols, there will be some complications with outputting HLS streams. This is because those streaming protocols require 10-30 second buffering. The player will not start playback until its buffer is filled, so the first user who initiated playback would need to wait until the system is ready. The only data source that doesn't have this problem is another Flussonic server with HLS protocol. Flussonic Media Server uses its own extensions that allows for instant playback on iPhone.

You can specify the stream's lifetime after a client has disconnected:

```
ondemand ipcam1 {
  input rtsp://localhost:554/source;
  retry_limit 10;
  clients_timeout 20;
}
```

The config line above has the following meaning: make **no more than 10 attempts to reconnect** with a data source if the connection is lost; when the last client leaves, run keep fetching the stream for **no longer than 20 seconds**.

Streams playback

How to playback streams, learn in Video output.

Stream screenshots in JPEG

Flussonic Media Server can generate JPEG thumbnails of a streaming video. To use this feature, add the `thumbnails` option in the stream settings:

```
stream example {
  input fake://fake;
  thumbnails;
}
```

Alternatively, to reduce CPU usage, you can specify an URL where Flussonic Media Server can get JPEG thumbnails. Many IP cameras have a special URL with screenshots:

```
stream example {
  input rtsp://localhost:554/source;
```

```
  thumbnails url=http://examplehost:5000/snapshot;
}
```

You can find the screenshot URL in the documentation for you camera model.

The latest screenshot of a stream is available at `http://flussonic:80/example/preview.jpg`

An MJPEG screenshot of a stream is available at `http://flussonic:80/example/preview.mjpeg`

See also:

- Thumbnails for JPEG thumbnails.
- Video Thumbnails for resource-saving MP4 thumbnails.

Substituting a stream with a file

If a stream becomes unavailable, Flussonic can substitute it with a fallback video from a video file that you specify using `backup <VOD location>`. This works for any live streams, including published ones.

```
stream example {
  input tshttp://10.0.4.5:9000/channel/5;
  backup vod/bunny.mp4;
}
```

You need to specify the path to the fallback file relative to the VOD-location, for example `vod/backup.mp4`, where `vod` is the unique name of our VOD location. Do not use absolute paths for video files.

> **Note**
>
> If the original stream has no audio (for example, a stream from an IP camera), the substitute file must have no audio as well.

By default, the fallback file is not recorded to the archive and is not transcoded. However, you can configure it.

To learn more about different ways of using files as stream failover sources, see the section **Source Failover**.

How 'backup' is different from 'input file://'

Unlike source switching with the source `input file://<VOD location>`, when a fallback file is used, Flussonic technically does not switch to another source. This is especially useful for published streams to prevent numerous closings of a socket with a publishing client.

The fallback file specified in `backup <VOD location>` is not transcoded and not written to DVR, unless you configure otherwise. The file source `input file://<VOD location>` is always written to DVR.

When use `backup` instead of `input file://`:

- In case of poor connection with the client that publishes video, Flussonic continues to receive frames without interrupting the connection with the client. This allows the client to continue the publishing session without having to start it over each time the source was switched. When the published stream disappears, viewers see the fallback file and understand that the broadcast is not over yet.
- When all sources are unstable and Flussonic switches between them too often, it is better to show a fallback file. If you use a file as one of the sources, viewers will see any video only after timeouts pass for each of the troubled sources.
- If you write the main stream to DVR and do not want to write the file too in order to prevent the file from appearing in the archived video.
- Using options like timeout for the main stream and the fallback file, you can manage which source to show during a publication session.

Wildcards

Sometimes names of streams on a remote server are not known in advance, so you want to dynamically rewrite the source (`input`) URL based on the **requested** stream name. To do that, use the following template feature:

```
template nsk {
  prefix nsk;
  input rtsp://streamer:555/%s;
}
template ams {
  prefix ams;
  input hls://streamer:8081/%s/index.m3u8;
}
```

The `template` directive uses the `%s` pattern to replace a substring in the input URL. It means that the substring that succeeds the prefix specified in the template's `prefix` replaces the `%s` pattern in the source URL, i. e. a substring that goes prior to the prefix, including the prefix itself, is omitted and what is left applies to the input URL. *Flussonic* receives a request for one URL but requests a different URL from the streamer without a redirect.

For instance, if a client requests a stream over the following URL: `http://FLUSSONIC-IP/ams/ort/index.m3u8` , *Flussonic* delivers the stream from the source URL: `hls://streamer:8081/ort/index.m3u8` , so the `http://FLUSSONIC-IP/ams/` substring is removed, and the `ort/index.m3u8` substring is used in the source URL `input` .

How do I use it?

For instance, to broadcast the channels to various areas from different servers, i. e. one server for every area. It provides you with a quick and efficient way of content delivery to the viewer.

Recording video streams (DVR)

Flussonic Media Server has a built-in state-of-the-art stream recording system.

The stream archiver can record video, provide access to a particular video interval, export parts of the archive as MP4 files, clean up old archive files, and maintain ample free space on the storage disk.

To turn on the archiver, specify the `dvr` option in a stream settings:

```
stream foxlive {
  input tshttp://trancoder-5:9000/;
  dvr /storage 90% 5d;
}
```

For details, see archive management.

Time zone adjustment (Timeshift)

Flussonic Media Server can play the archive record of a stream with a fixed delay.

> **Warning**
>
> Flussonic Media Server maintains the exact delay, so if for some reason the archive has gaps, end users will be getting no video for the duration of a gap.

The timeshift feature has its own data source schema — `timeshift://` :

```
stream channel {
  input fake://fake;
  dvr /storage 90% 5d;
}
stream channel-2h {
  input timeshift://channel/7200;
}
```

The delay is specified in seconds.

Stream delivery over UDP multicast

Flussonic Media Server can rebroadcast a stream from a data source over the local network.

Flussonic Media Server demonstrates next-to-ideal jitter values when streaming multicast UDP over the network.

```
stream example_stream {
  input tshttp://localhost:80/origin/mpegts;
  push udp://239.0.4.4:1234;
}
```

Stream settings for IP surveillance cameras

It is possible to configure Flussonic Media Server to request a stream from camera via UDP only. This might be useful when dealing with cameras that have issues with streaming over TCP.

```
stream cam1 {
  input rtsp://localhost:553/bunny.mp4;
  rtp udp;
}
```

> **Warning**
>
> The HEVC (H.265) video codec is supported **only** in Chrome (version 107 and higher), Microsoft Edge (version 16 and higher), and Safari (version 11 and higher) on desktops, and in Chrome (version 107 and higher) and Safari for iOS (version 11.0 and higher) on mobile devices. All other browsers cannot play H.265 video streams. More on this in Playing H265.

If there is no need to retrieve audio from a camera (for example if the audio is encoded in G.726), you can configure Flussonic Media Server to ingest only one track. The number of the track must be specified in the stream settings:

```
stream cam1 {
  input rtsp://localhost:554/origin tracks=1;
}
```

In order to transcode an audio stream from G.711a or G.711u into the AAC codec, use the protocol `rtsp2`:

```
stream cam1 {
  input rtsp2://localhost:554/origin;
}
```

Turning on audio-only HLS

When approving apps for publishing in AppStore, Apple may require the stream to have an audio-only version. To satisfy this requirement, add `add_audio_only` directive to the configuration:

```
stream cam1 {
  input fake://fake;
  add_audio_only;
}
```

With this directive, if the stream contains both audio and video, Flussonic will generate multibitrate playlist with two profiles — one with audio only and another with audio and video tracks.

Capturing stream from another Flussonic Media Server

The details of transferring video between Flussonic Media Server servers are discussed in Flussonic video stream clusterization.

DRM in live streaming

The details are discussed in the DRM article.

Silence detection

Flussonic can detect low sound level (no sound) in sources of input streams and notify about it. See Silence detection for details.

3.2.4 Publishing video to the server

Flussonic Media Server can accept video from external systems and devices that initiate broadcast. This is called *publishing* to *Flussonic*.

Publishing can be used in a situation where the external system has no static IP or where it is located behind the firewall in a private IP network. In this case, *Flussonic* has no way of directly sending a request for video.

WHAT WE CALL *PUBLISHING* TO FLUSSONIC:

- Transmitting video from a mobile device to *Flussonic*.
- Transmitting video from OBS (Open Broadcaster Software) or vMix to *Flussonic*. Learn more
- Transmitting video from a webpage to *Flussonic* via WebRTC. Learn more

WHAT WE DON'T CALL *PUBLISHING* TO FLUSSONIC:

- Receiving a multicast
- Ingesting a stream from some source

In those cases, *Flussonic Media Server* has to connect to the data source. Whereas the case where the connection is not initiated by *Flussonic* itself is called *publishing*. For example, publishing is when a mobile device connects to *Flussonic* to transmit video.

Publishing video to social networks is not publishing to *Flussonic* and therefore it does not meet *Flussonic*'s definition of publishing as used in this documentation.

SUPPORTED PROTOCOLS

Flussonic Media Server can receive requests for video publishing via the RTMP, RTSP, HTTP MPEG-TS, WebRTC, and SRT protocols.

CONTENTS

Publishing to a static stream

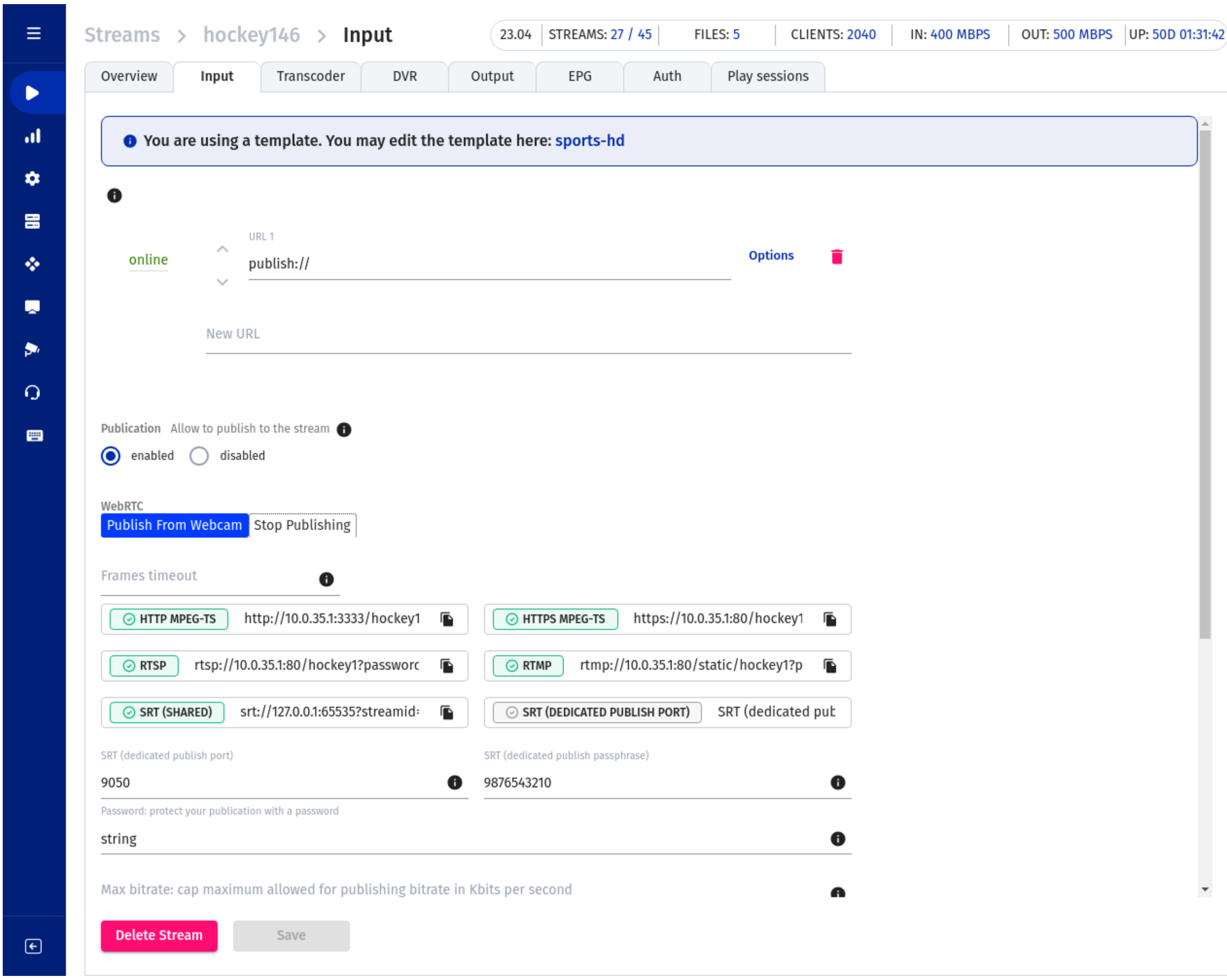
If you know what stream name an external system will use to publish video to *Flussonic*, you can create a stream with that name and a `publish://` data source.

To create a static stream with a publishing source:

1. In the admin interface, create a stream: **Media** > **Stream** > **add**.
2. Enter a **Stream name**.
3. Specify `publish://` as the **Source URL**. Alternatively, save the stream, go to the **Input** tab, and set the **Publication** switch to **enabled**.

> **Note**
>
> To disable publication to the stream, delete the `publish://` input or set the **Publication** switch to **disabled**.

4. Click **Create** to save the stream.
5. To specify additional options for a published source, click **options** next to the URL.



This setting corresponds to the following line in the configuration file:

```
stream published {
  input publish://;
}
```

URLS FOR PUBLISHING VIA VARIOUS PROTOCOLS

To view URLs for publishing to a specific stream:

1. Click the stream name in **Media** and go to the **Input** tab.
2. All available URLs for publishing will be displayed in the **Publish links** section (see the screenshot above).

You can publish videos to *Flussonic* using the following URLs:

- **RTSP**: `rtsp://FLUSSONIC-IP/stream_name`
- **HTTP MPEG-TS**: `http://FLUSSONIC-IP/stream_name/mpegts`
- **RTMP**: `rtmp://flussonic-ip/published` or `rtmp://flussonic-ip/static/published`. Learn more about RTMP URL at Publishing RTMP streams.
- **WebRTC**: `http://FLUSSONIC-IP/stream_name/whip`. Learn more about publishing via WebRTC at Publishing SRT streams.
- **SRT**: `srt://FLUSSONIC-IP:SRT_PORT?streamid=#!::r=STREAM_NAME,m=publish`. Learn more about SRT URL at Publishing SRT streams.

Publishing with a dynamic name

WHY USE DYNAMIC NAMES AND PUBLISHING LOCATIONS?

You might want to use dynamic names for published streams if one or more of the following is true:

- Your publications last for a limited period of time (unlike a 24/7 TV channel broadcast).
- You manage a lot of publications, and it is too much work to create a separate stream for each of them.
- You do not know the names of the incoming streams in advance. For example, you are dealing with a third-party application – like a web chat – that generates a new unique identifier for each stream that it publishes to *Flussonic*.

Flussonic solves these problems by allowing you to create a *publishing location* (*publication prefix*) where you can specify common settings for multiple streams.

A **dynamic name** means that the full name of a stream is formed from a pre-configured *publication prefix* and the name defined in an external app.

If the name of a published stream is not known beforehand, or if you expect many published streams, you should set up a *publication prefix*:

```
template chats {
  prefix chats;
  input publish://;
}
```

Here, `chats` is the publication prefix.

All streams published under the `chats` prefix will have settings that you specify in the `template` directive. To learn more about stream settings and options, refer to the Flussonic Media Server API.

You can specify several prefixes in the `template` directive to create several publishing locations. You can also specify an empty prefix (`""`) to publish a stream with any prefix or even without a prefix. Learn more at Templates and prefixes.

URLS FOR PUBLISHING VIA DIFFERENT PROTOCOLS

In case of publishing with a dynamic name, you will need to publish streams under names with a prefix, for example:

- **RTSP**: `rtsp://FLUSSONIC-IP/template_name/stream_name`
- **HTTP MPEG-TS**: `http://FLUSSONIC-IP/template_name/stream_name/mpegts`
- **RTMP**: `rtmp://FLUSSONIC-IP/template_name/stream_name` . Learn more about RTMP URL at Publishing RTMP streams.
- **SRT**: `srt://FLUSSONIC-IP:SRT_PORT?streamid=#!::m=publish` . Publishing via SRT with a dynamic name is only supported for this URL format, i.e. when separate port for a stream or a group of streams is set.
- **WebRTC**: `http://FLUSSONIC-IP:PORT/template_name/stream_name/whip` .

The part of the name that goes after `template_name` is defined in the client app. *Flussonic Media Server* does not "know" the stream name in advance.

Testing a publication

PUBLISHING VIA RTMP

To test that publishing over RTMP works, you can use `ffmpeg` :

```
ffmpeg -re -i /opt/flussonic/priv/bunny.mp4 -vcodec copy -acodec copy -f flv rtmp://localhost/chats/tmp
```

This command should cause a new stream to appear in the web interface:

Learn more about RTMP publishing URL at Publishing RTMP streams.

PUBLISHING VIA RTSP

Some clients can publish video over RTSP.

Flussonic Media Server supports automatic selection between UDP and TCP transport and will receive the stream using the protocol selected by the client.

The stream name must be complete: `chats/my/chat-15`

```
ffmpeg -re -i /opt/flussonic/priv/bunny.mp4 -vcodec copy -acodec copy -f rtsp rtsp://localhost/chats/my/chat-15
```

PUBLISHING VIA MPEG-TS

When transcoding a stream using `ffmpeg`, it is possible to publish video over HTTP. Video can be published with `mpegts` added at the end of the URL:

```
ffmpeg -re -i /opt/flussonic/priv/bunny.mp4 -vcodec copy -vbsf h264_mp4toannexb -acodec copy -f mpegts http://localhost:80/chats/my/chat-15/mpegts
```

PUBLISHING VIA SRT

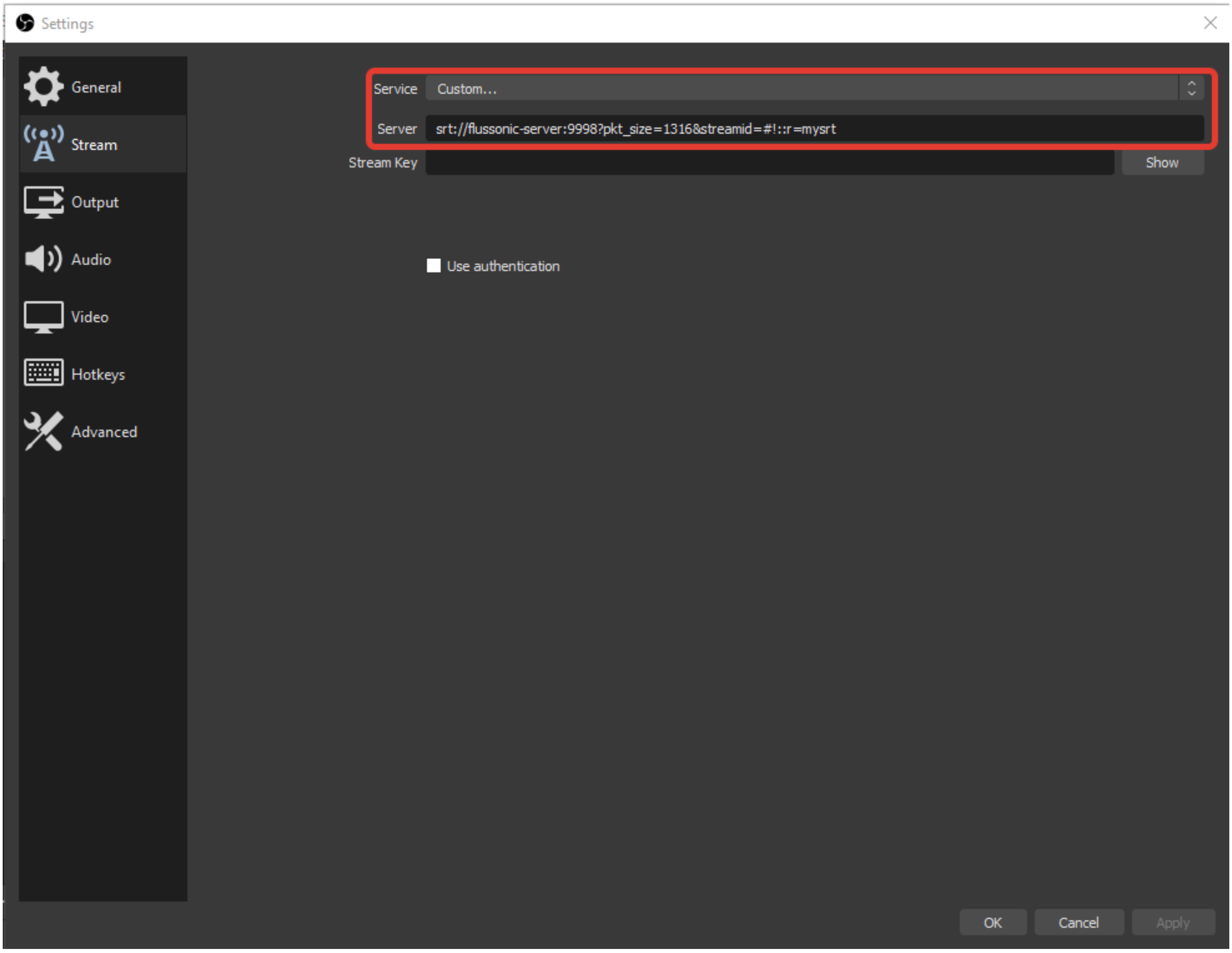
You can use `ffmpeg` to test the publishing:

```
/opt/flussonic/bin/ffmpeg -re -i PATH_TO_VIDEO -c copy -y -f mpegts 'srt://FLUSSONIC-IP:SRT_PORT?pkt_size=1316&streamid=#!::r=STREAM_NAME,m=publish'
```

, where:

- `FLUSSONIC-IP` is the *Flussonic* IP address.
- `SRT_PORT` is an SRT port.
- `STREAM_NAME` is the stream name.
- `m=publish` is a publishing mode.

Or, you can publish a stream from any other 3d party software supporting this protocol, for example OBS Studio:



PUBLISHING VIA WEBRTC

To test a WebRTC publication, you can publish video from your webcam or use our demo app.

Protecting a publication with a password

Flussonic Media Server can verify a password when publishing a stream. Enter the password in the configuration file as follows:

```
template chats {
    prefix chats;
    password mypass;
    input publish://;
}
stream published {
    password secure;
    input publish://;
}
```

Examples for testing:

- To publish a password-protected **RTMP** stream, use the following example:

  `rtmp application rtmp://192.168.2.3/live`

  `stream name mystream?password=mypass`

- To publish an **HTTP MPEG-TS** stream, you can enter the data as follows:

```
http://192.168.2.3:80/s1/mpegts?password=secure
ffmpeg -re -i video.mp4 -vcodec copy -acodec copy -f flv rtmp://192.168.2.3/live/mystream?password=mypass
ffmpeg -re -i video.mp4 -vcodec copy -bsf h264_mp4toannexb -acodec copy -f mpegts http://192.168.2.3:80/s1?password=secure
```

Authorization of a publishing source

Flussonic Media Server allows you to configure an HTTP handler that will check additional information about the publisher (that is, the source of a published stream) before accepting or rejecting the stream. Learn more at Publishing session authorization.

DVR archives and dynamic names of streams

You can configure the DVR archive for a publishing prefix:

```
template recorded {
  prefix recorded;
  input publish://;
  dvr /storage 3d 500G;
}
```

In this case, the published video will be recorded, and will be available even if the publication is terminated.

When the client stops publishing the video, the stream will disappear after some time, and *Flussonic Media Server* will keep a very little information about it. Information about this stream will be stored in the index of the archive, and *Flussonic Media Server* will not lose the files on the disk.

If configured, the system of purging of the archive will delete published streams according to the schedule.

Republishing

To republish the streams, use `push` with a template (`%s`):

> **Danger**
>
> We do **not** recommend using `push` over UDP in this case as it causes a collision.

```
template pushed {
  prefix pushed;
  input publish://;
  push rtmp://CDN-SERVER:1936/mylive/%s;
}
```

With the configuration above, *Flussonic* republishes the `pushed/mystream` stream, using the following URL: `rtmp://CDN-SERVER:1936/mylive/mystream`.

Switching stream sources, using timeout

The rules of switching sources according to their priority and state (whether a source is available or not) apply to published sources too. This means that you can add alternative sources to a stream with a published source and use timeout for switching between sources.

If a published source is unavailable, Flussonic immediately switches by default to the next source. Alternatively, you can specify your custom timeout.

Example with multiple sources and a timeout:

```
stream published {
    source_timeout 3;
```

```
    input publish://;
    input file://vod/bunny.mp4;
}
```

You can also specify timeout for each source individually:

```
stream published {
    input publish:// source_timeout=3;
    input file://vod/bunny.mp4 source_timeout=2;
}
```

PROHIBITING THE PUBLISHING, THE PUBLISH_FORBIDDEN EVENT IN LOGS

When the source `input publish://` has a lower priority than the other specified stream sources, it might mean that the publication will not actually happen. You can prevent the publication and allow it again by changing the priority of sources. This can be done during the broadcast.

If publishing is not possible, Flussonic generates the event `publish_forbidden`. For example, this event occurs with the following configuration if the file `bunny.mp4` exists and is successfully played:

```
stream published {
    source_timeout 3;
    input file://vod/bunny.mp4;
    input publish://;
}
```

To allow publication, put the source of publication before the file.

3.2.5 HLS inputs on-demand

Contents

What are on-demand streams for?

If the stream is set to **On-demand**, it will only turn on when it receives a request from the user. If the user has lost the need for viewing, or the source from which the stream is broadcasted has stopped its work for some reason, the stream will automatically turn off after a certain time (timeout). On-demand is mainly needed to save resources (primarily traffic). This solution is well suited in situations where you have a large number of streams, but you do not need them to work simultaneously and continuously.

Read more about setting up On-demand streams here.

Problems with on-demand streams

The peculiarity of segmented protocols in On-demand is that when using them, the media player's buffer must be accumulated before the start of video playback (initial loading of at least 3 video segments), which takes a certain amount of time. Although parameters such as specific communication protocol, GOP size, segment length can affect the buffer accumulation time, this process and the associated delay are inevitable.

Problems with HLS protocol

1. HLS is a segmented protocol, so it takes a certain amount of time to accumulate a buffer.
2. In Flussonic Media Server, a check for determining the liveness of a source is implemented. Flussonic (restreamer), while connected to the HLS source, remembers the last seen segment and waits for the next one to appear. Flussonic will not start playing until it appears. This introduces additional playback delay.

> **Note**
>
> In case we receive an unknown (non-Flussonic) HLS source and relay it, without this check **there is no other way to know** whether the source is online or not. The absence of this check can lead to a infinite playback loop of the last read segments if the source goes offline at some point. Flussonic cannot trust a third-party source (even if it provides the required data), because it is impossible to determine what this source is.

The combination of all these factors will lead to the fact that you will face the problem of a stream's long start time.

How to fix it?

We have a solution for both of these problems — our proprietary `M4F` protocol. Its use **removes the need to check the source** (this protocol ensures that restreamer does not load already existing segments a second time, and source will reset the playlist if frames are missing on it), it also supports **prepush** (fast filling of media player's buffer) and provides information about archive of the source (or several sources at the same time), makes it possible to proxy the archive . As a result, both reasons for this delay are resolved. `M4F` is an internal Flussonic protocol, therefore, its use is possible only if Flussonic Media Server is both a source and a restreamer. Learn more about the `M4F` protocol.

> **Note**
>
> If your primary goal is On-demand streams and their output via HLS protocol, we strongly recommend you to contact your supplier with a request to install a Flussonic Media Server, and use the `M4F` protocol to connect between them. Eventually, you will get a well-synchronized configuration and solve the problems associated with long stream startups.

3.2.6 Multicast source redundancy

An IPTV service based on multicast requires a multicast source redundancy. When the primary multicast source is unavailable, the system should switch to an active one. Depending on whether you are a content provider or a client depends on what goal you are achieving:

- Source redundancy when receiving a multicast stream.
- Source redundancy when delivering a multicast stream (TwinCast module of Mcaster).

Source redundancy when receiving a multicast stream

Several content providers send the same multicast content to the IXP (Internet Exchange Point), where dozens of operators receive it. Various errors may occur in such a system, such as data duplication or excessive traffic on the receiver. The IXP (Internet Exchange Point) owners have to manage multicast groups so that an operator could receive the required channel.

The IXP (Internet Exchange Point) owners can manage multicast groups in the following ways:

- Separate providers into different multicast groups: provider-to-multicast group.

  Allocate a separate multicast group for each content provider and keep a single register of multicast groups. This way, providers will broadcast to the separate multicast group addresses. To receive a single channel from a provider, an operator has to filter all the incoming content on the receiver, receiving excessive traffic. For operators to find sources of the same content to achieve redundancy, they have to search for channel duplicates in the multicast group register.
- Separate channels into different multicast groups: channel-to-multicast group.

  Allocate a separate multicast group for each channel. Content providers will send each channel to its separate multicast group, causing data duplication and confusion at the IXP (Internet Exchange Point). So operators have two ways to connect to the particular source and receive traffic:

  ```
  - Filter traffic by source IP address at the receiver. This way, the receiver server will receive excessive traffic.

  - Implement SSM (Source-Specific Multicast) with IGMPv3. When connecting to the IXP (Internet Exchange Point), the operator specifies the multicast group
  address and the channel source to receive the data. If the requested source is unavailable or offline, the receiver will automatically switch to another
  active source (see [how SSM works](#redundant-multicast-streaming-receive-ssm-work)). For SSM to work, the operator should have the source address in
  advance.
  The SSM (Source-Specific Multicast) mechanism simplifies the management of multicast groups without creating additional issues at the IXP (Internet
  Exchange Point) or at the receiver.
  ```

SSM (Source-Specific Multicast) mechanism achieves the following goals:

- Multicast source redundancy. SSM supports multiple sources simultaneously broadcasting to the same multicast group. So, if two providers send content to the same multicast group address, without SSM (Source-Specific Multicast) it results in duplicated data and confusion at the receiver, but with SSM (Source-Specific Multicast) it's a standard situation.
- Receiving a multicast stream from a provider that requires compliance with the standard of multicast stream delivery. Some content providers require the operator's equipment to support SSM (Source-Specific Multicast) and IGMPv3 to receive multicast.

HOW DOES SSM WORK

> **Note**
>
> SSM mechanism requires a switch to support IGMPv3.

SSM works as follows:

1. Each multicast packet transmits the source IP address in the header. The primary and backup sources simultaneously send data to the same multicast group address.
2. Using the source IP address, Flussonic calls the switch via IGMPv3 to send the multicast packets from the primary source.
3. If Flussonic detects packet loss from the primary UDP source within source_timeout that is by default one minute, it switches to the backup source. Using the backup source IP address, Flussonic calls the switch via IGMPv3 to send the multicast packets from the backup source.
4. While on the backup source, Flussonic polls the primary UDP source every minute for multicast packets.
5. When the primary source comes back online and starts sending packets, Flussonic switches to it from the backup source.

Diagram 1. SSM with the online primary source

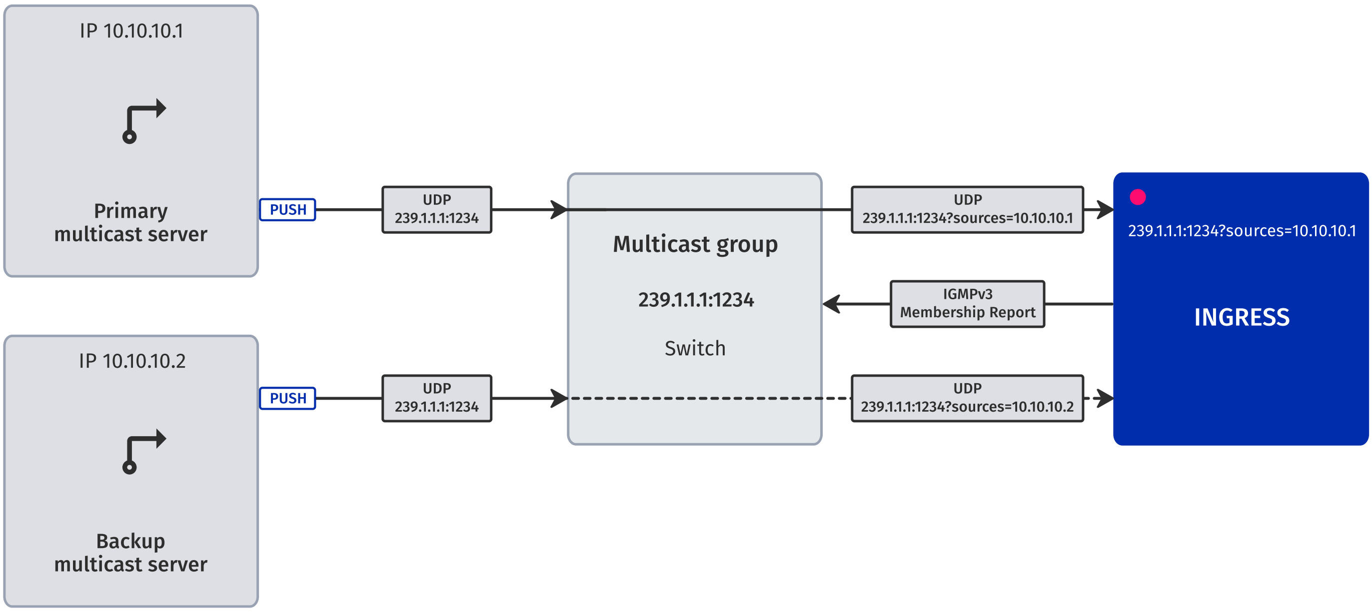
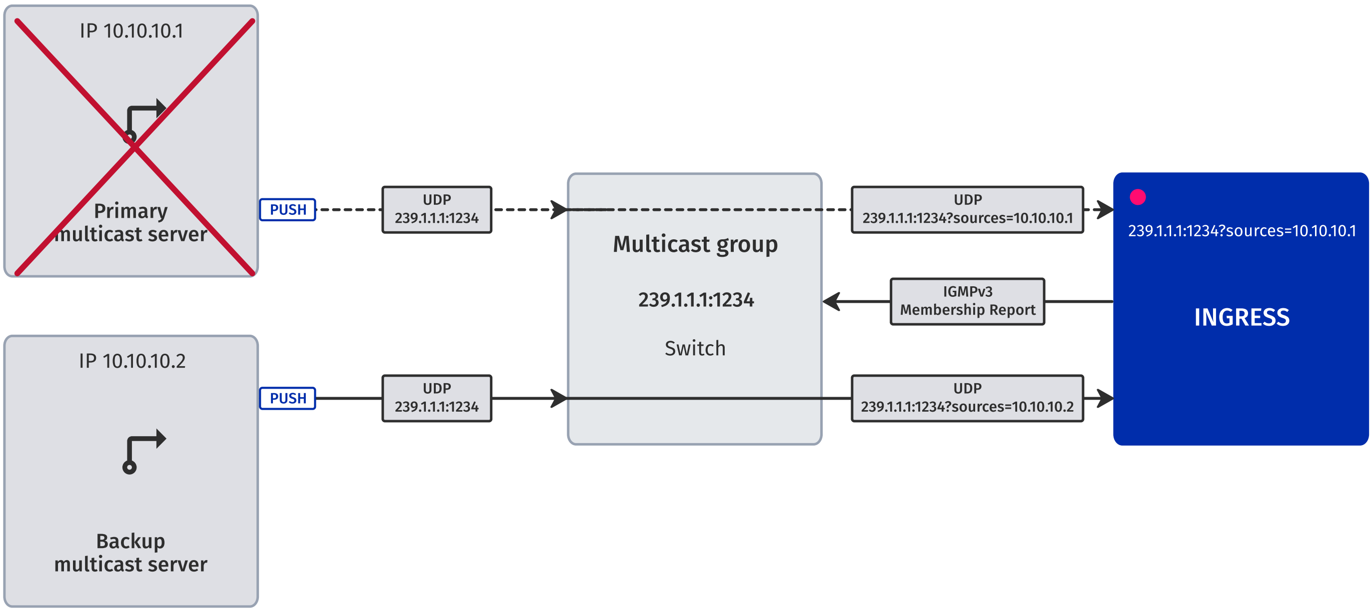


Diagram 2. SSM with the offline primary source and online backup source



To configure the SSM (Source-Specific Multicast) mechanism for the multicast stream in the Flussonic Admin UI, follow these steps:

1. Go to the **Input** tab of the stream settings.
2. Add a primary source if you don't have one or modify the existing one by specifying the multicast group address, port, and IP address of the primary source like so:

   `udp://239.1.1.1:1234?sources=10.10.10.1`

   , where:

   - `239.1.1.1.1` is the multicast group address
   - `1234` is the port for Flussonic to listen to
   - `10.10.10.10.1` is the source IP address
3. Add a backup source by specifying the data as in step 2 for the backup source.
4. Apply the settings by clicking **Save**.

3.2.7 How to create your own IPTV channel (server-side playlist)

On the page:

- Overview of server-side playlist in Flussonic.
- How to create an IPTV channel from video files.
- How to create an IPTV channel from streams.
- Setting up the streaming schedule.
- How to configure ad insertion for server-side playlist.

As an IPTV operator, you can create a **FAST (Free Ad-Supported streaming TV)** channel—free to viewer channel with ads. For example, a movie channel or a barking channel—a channel dedicated to informing your subscribers and advertising new products and services. With Flussonic, you can create such a channel by assembling VOD files and live content into playout streams.

FAST channel with video files from local VOD location instead of streams received over the network helps to:

- Save the web traffic.
- Attract new subscribers. Viewers have free access to watch such a channel.

> **Note**
>
> Make sure that you aren't violating intellectual property rights.

Overview of server-side playlist in Flussonic

A custom channel is a playlist that contains links to sources, such as video files and streams on Flussonic Media Server. A custom playlist plays specified sources on a loop and can run on a schedule.

Server-side playlists can be used to:

- Broadcasting a TV channel to many simultaneous viewers in a local network.
- Switch between several streams. For example, you can create a playlist that switches between CCTV camera streams every other minute.
- Create a digital signage platform to display informational clips or commercials.

DISADVANTAGES OF SERVER-SIDE PLAYLISTS

Server-side playlists have several disadvantages when used on the Internet to embed video into websites:

- You can't use targeting when inserting ads.
- You can't use AdRiver and other similar systems for ad analytics.
- Complexity of creating a multi-bitrate broadcasting: different files can contain different number of tracks of different bitrates.
- Rewinding, one of the major advantages of online streaming, is difficult to implement.
- The pause function is complex to implement.

The main disadvantage of such playlists is that they offer no means to create an adequate ad analytics system. Instead of server-side playlists, it's recommended to use client-side playlists. These playlists allow an IPTV subscriber to select channels and form a playlist.

However, server-side playlists can be used for purposes other than online broadcasting. Practice shows that users are more willing to watch predefined content rather than to search for videos themselves.

How to create an IPTV channel from video files

To create a server-side playlist from video files, follow these steps:

1) Prepare video files for further live streaming.
2) Set up a VOD location and upload the video files.
3) Create a server-side playlist.
4) Create a stream with the `playlist://` source.

STEP 1. PREPARE VIDEO FILES

To ensure compatibility across devices and optimal playback, prepare video files. Video files in the playlist should have identical parameters, such as audio and video codecs, resolution, and bitrate.

STEP 2. SET UP A VOD LOCATION

1) The default directory for video files is `/opt/flussonic/priv`. It already exists in the configuration file `/etc/flussonic/flussonic.conf`:

```
vod vod {
storage /opt/flussonic/priv;
}
```

or:

```
vod vod {
storage priv;
}
```

The example uses the default directory that's specified in `vod`. If you want to use another directory, create another VOD location or add the path to the existing VOD location. You can specify storage for playlist's files using the Flussonic UI.

1) Upload video files in the specified directory.

The example uses `bunny.mp4` and `beepbop.mp4`, which already exist in the `/opt/flussonic/priv/`.

STEP 3. CREATE A SERVER-SIDE PLAYLIST

A playlist is a text file with a list of links to sources. To edit the playlist, you can use nano—a text editor for Linux systems.

1) Install nano by running these commands:

`apt-get update && apt-get install nano`

2) In the directory with the video files, create a file `playlist.txt` by using the following command:

`nano /opt/flussonic/priv/playlist.txt`

The editor immediately opens the file. Add the links to video files for streaming. The link consists of the name of the VOD location and the video file name:

```
vod/bunny.mp4
vod/beepbop.mp4
```

To exit and save the changes, press **CTRL + X** and then **y**.

STEP 3. CREATE A STREAM WITH THE `playlist://` SOURCE

1) Create a static stream by going to the **Media** section in the navigation menu to the left and clicking **+** next to the **Streams** tab. Specify the stream name and the source URL. The source URL consists of the `playlist://` setting and the path to the playlist file. For example, `playlist:///opt/flussonic/priv/playlist.txt`, where `/opt/flussonic/priv/playlist.txt` is the path to the playlist file.

You can also create a stream in the configuration file `/etc/flussonic/flussonic.conf` or with an API request `Flussonic-API: PUT /streams/{name}`.

2) If you edit the configuration file, reload the server configuration by running the following command in the Linux command line:

`service flussonic reload`

A new stream will appear in the list of streams in the web interface. The stream will play the specified files on a loop.

How to create an IPTV channel from streams

A server-side playlist from streams is similar to a playlist from video files except that you specify streams instead of VOD files.

To create a server-side playlist from streams, follow these steps:

1) In the VOD location directory, create a playlist `playlist.txt`, and specify the names of the streams on the Flussonic server to broadcast. To manage the broadcasting schedule, use the tags. See the example playlist `playlist.txt`:

```
#EXTINF:60
cam1
#EXTINF:60
cam2
```

where:

- `cam1` and `cam2` are the stream names
- `#EXTINF:60` is the tag that specifies the duration in seconds to play the stream

The playlist plays streams `cam1` and `cam2` sequentially, switching between the streams every 60 seconds.

2) In the Flussonic UI create a stream with `playlist://` source, specifying the path to the playlist. Click **Save** to apply the settings:

You can also create a stream in the configuration file `/etc/flussonic/flussonic.conf` or with an API request `Flussonic-API: PUT /streams/{name}`.

Setting up the schedule

To set up a schedule in a playlist, use the following tags:

- `#EXT-X-MEDIA-SEQUENCE`. The serial number of the first element in the playlist. You can use it to correctly rotate through and update a playlist.
- `#EXTINF`. The duration in seconds to play a playlist element. You can use it to embed live content into a playlist.
- `#EXT-X-UTC`. The start time in UTC of a playlist element.
- `#EXT-X-PROGRAM-DATE-TIME`. The start time in ISO 8601 format of a playlist element: `2013-02-12T12:58:38Z` (GMT).
- `#EXT-X-CUE-OUT:ID=SOME_ID`. The ad marker represents the splice start point.
- `#EXT-X-CUE-IN`. The ad marker represents the splice end point.

Find the examples of applying tags in the Examples section.

> **Note**
>
> Every time a file in a playlist finishes playing, Flussonic re-reads the playlist.

Consider the following rules for processing playlists:

1) If you specify the tag `#EXT-X-MEDIA-SEQUENCE`, the playlist remembers the last played element, and playback continues from the next element after re-reading. The playlist will be synced from the next number. If the new playlist has only numbers less than the last number, the playlist file will be re-read every second, waiting for the correct number.

2) If the tag `#EXT-X-MEDIA-SEQUENCE` isn't specified and the playlist file hasn't been changed, then the next element will be played. If the file has been changed, playback starts from the beginning.

EXAMPLES

- With the `#EXTINF` tag, you can set the playback duration for each playlist item. For example, broadcast the first 30 seconds of the first file and the first 60 seconds of the second file:

```
#EXTINF:30
vod/bunny.mp4
```

```
#EXTINF:60
vod/beepbop.mp4
```

- With the tag `#EXT-X-UTC`, you can set the time in UTC when you want to play the playlist element:

```
#EXT-X-UTC:1522839600
vod/bunny.mp4
#EXT-X-UTC:1522843200
vod/beepbop.mp4
```

- With the `#EXT-X-PROGRAM-DATE-TIME` tag, you can set the start time in the ISO 8601 format of the playlist element:

```
#EXT-X-PROGRAM-DATE-TIME:2018-04-04T11:00:00Z
vod/bunny.mp4
#EXT-X-PROGRAM-DATE-TIME:2018-02-04T12:00:00Z
vod/beepbop.mp4
```

How to configure ad insertion for server-side playlist

The process of creating a server-side playlist to insert ads into a stream without ad markers is similar to creating a playlist from video files except for the following:

1) You should prepare both video files and ad clips and upload them to the VOD location.

> **Warning**
>
> Video files with content and ad clips in the server playlist must have the same video and audio codecs, resolution, and bitrate. Otherwise, the players won't handle it correctly.

2) The playlist name must end in `.onlyvod.m3u8`. For example, `ad_playlist.onlyvod.m3u8`. This enables the ad insertion mechanism.

3) The playlist must contain the splice start (`#EXT-X-CUE-OUT:ID=SOME_ID`) and end (`#EXT-X-CUE-IN`) point markers for the ad insertion. Playlist example:

```
vod/film_1.mp4
#EXT-X-CUE-OUT:ID=126
vod/ad_1.mp4
vod/ad_2.mp4
#EXT-X-CUE-IN
vod/film_2.mp4
#EXT-X-CUE-OUT:ID=127
vod/ad_3.mp4
#EXT-X-CUE-IN
vod/film_3.mp4
```

In HLS and DASH playlists, you'll see SCTE markers of when ads start, for example:

```
#EXT-OATCLS-SCTE35:/DAgAAAAAAAAAP/wDwUAAAB+f/8AABdmoAABAAAAAG1gDGA=
#EXT-X-CUE-OUT:DURATION=17.040
```

See also:

- Overlaying the logo
- Make your channel available over UDP in the local network

3.2.8 Creating TV channel from IP Camera

This article will explain how to add an IP camera to *Flussonic* and then add it to an MPTS stream broadcasted to a cable or satellite network. Such a value added feature may become a pleasant bonus for your viewers; for example, you can use it when deploying a TV broadcast in a gated community to display a public camera surveilling the gates as a separate channel, or broadcast the process of construction of some important infrastructure facility. If desired, you can give viewers access to the camera archive.

To add a camera to the multiplexer:

1. Find out the camera's RTSP URL.
2. Create a stream with this RTSP URL in *Flussonic*.
3. Add the created stream to a multiplexer.

The instructions on those are given below on this page.

RTSP URL

Flussonic allows you to connect any cameras supporting RTSP so you have to find out the RTSP URL of your camera. Usually you can find it in the camera's web interface. Pay attention to the following points:

- The URL must contain login and password
- You need the IP address of the camera to be accessible from *Flussonic*.

Normally RTSP URL looks like this: `rtsp://admin:4321@192.168.45.32/cam/realmonitor?channel=1&subtype=1`. It's important that usually there's some path after the IP address. Your camera won't stream without it.

Some cameras include a login and password in the URL, and then it takes the form of `rtsp://192.168.0.213/user=admin_password=tlJatbo6_channel=1_stream=0.sdp?real_stream`.

Sometimes when the camera is in a closed network, it's necessary to forward ports on a router. In this case, the IP address and port of the web interface are not equal to the IP address and port of your camera. Some cameras handle this situation incorrectly, and may offer you an RTSP URL that contains an internal IP address. In this case, it's necessary to replace the address and port with external ones.

So, to sum up, please note that the IP address of the camera isn't enough for connection with *Flussonic*, you must specify a correct and accessible RTSP URL.

Creating a new stream

Now you need to create a new stream in *Flussonic*.

1. Go to **Media** — **Streams** and click **+**.
2. In the stream creation form, specify the stream name name and the RTSP URL that is described in the section above.
3. Click **Create**.
4. The profile of the created stream opens on the **Overview** tab. If the URL is correct, you will see statistics of bitrate and tracks, as well as the stream preview in the player, etc.
5. If necessary, you can configure the RTSP-specific settings of the stream. To do so, go to the **Input** tab ang click **Options** next to the input URL. RTSP settings include:
    - **RTP transport type**: select TCP or UDP transport protocol.
    - **RTSP tracks**: enter specific tracks of the RTSP stream that are to be captured.
    - **Output audio**: select the output audio codec if the source one does not play.
6. To enable the archive recording, go to the **DVR** tab in the stream profile.

See also:

- Examples of configuration when adding an RTSP stream in the configuration file.
- Alternative ways of using the stream from camera — adding video to websites (embed.html), video screenshots.

Adding the stream from the camera to a multiplexer

Adding the stream from the camera to a multiplexer is not different from adding any other stream. This can be done in the UI:

1. Go to **Media** — **Multiplexers**.
2. Find the required multiplexer or create a new one.
3. In the **Add program** field, enter or select the name of the camera stream that you have created earlier.
4. Click **+**.

Done, now the video from the camera will be added to the multiplexer. Mandatory settings such as program number (PNR), track number (PID), etc. are set automatically. If necessary, you can expand the settings and set all parameters of the program and the multiplexer, see article about configuring multiplexer.

3.2.9 Processing audio from IP Cameras

Processing audio from IP cameras

Most IP cameras support only PCMA/PCMU audio codecs (also known as G.711a and G.711u).

Flussonic can store this PCMA/PCMU audio in the archive and transmit it over RTMP that supports this codec. Due to the infeasibility of transmitting such audio over other protocols such as HLS or storing it in MP4 for a playback, client simply will not hear the sound as these codecs are not supported by HLS or MP4.

Therefore, in order to make the audio available for playback to all client's devices and players, enable transcoding of the audio received from an IP camera.

To do so, first install the `flussonic-transcoder` package:

```
apt-get install -y flussonic-transcoder
```

Then enable sound transcoding for your stream from the camera as follows:

```
stream camera1 {
 input rtsp://localhost:554/origin output_audio=aac;
}
```

This way *Flussonic* will transcode the audio track to AAC codec, so it will be possible to send it to all of your clients over any protocols without the loss of a sound.

3.2.10 NDI Source Capture

Flussonic supports video capture using the NDI protocol.

Usually you will be provided by following NDI URL: `ndi://Server_name (Source 1)`.

Convert it to `ndi://Server_name/Source 1` because this must be an URL.

The NDI codec is not compatible with browsers, so to get HLS or send it to another server, transcoder activation is required.

```
apt install -y ndi-bridge
```

3.2.11 Ingest multicast

Flussonic Media Server can ingest multicast over UDP MPEG-TS, including RTP encapsulation.

In computer networks, multicast sent from a source stops at the first switch, and doesn't proceed further until a client in the network explicitly requests to join the multicast group to receive the data. To receive multicast OS kernel sends an IGMP request to the switch. While the software is running, the kernel resends IGMP requests to the switch, extending the validity of the data request to join the multicast group. Without IGMP requests to the switch, the client will stop receiving multicast in a few seconds.

Contents:

Requirements

- A server with Flussonic Media Server installed. A virtual server might not suit due to the specific multicast operation and high complexity of network configuration.
- A source of unencrypted multicast. You should know its multicast group address and port. You can start with SPTS source, that's one channel—one group. Then you can move on to MPTS and T2-MI encapsulation.

Ingesting SPTS from a source using a server with a single interface

If your server has a single network interface, follow these steps:

1. Create a stream with some name.
2. Specify the source like so: `udp://239.0.0.1:1234`.

```
stream example {
    input udp://239.0.0.1:1234;
}
```

When playing video and audio in a browser, you may encounter the following issues:

- If you have H.264 (AVC) or H.265 (HEVC) video, the browser will play it but not on all Apple smartphones. If you have an MPEG-2 video, you'll need specialized software, like VLC to play the video.
- Multicast transmits MPEG-2 or AC-3 audio. Browser can't play either of them.

It's enough to verify the following:

- The stream is active.
- Stream's uptime grows without any interruptions,
- The input bitrate matches the expected bitrate, varying from 1 to 10 Mbits:

Selecting a network interface

A server receiving multicast usually has more than one network interface. It may have one network card connected to LAN and used to receive video, and the other one connected to the Internet (WAN) and used to serve clients via HLS or HTTP MPEGTS and download updates.

The default route uses WAN interface, so the OS kernel sends IGMP requests to the WAN interface. It means that Flussonic Media Server won't receive the video.

To explicitly specify which interface to use to request and receive multicast, specify the name of the interface, for example, `eth2` in the source IP address right before the multicast group address:

```
stream example {
  input udp://eth2@239.0.0.1:1234;
}
```

If it's more convenient for you to specify the IP address of the interface to which to send the IGMP request, then specify it in the source address right after the address of the multicast group. For example, if the `eth2` interface has the `10.100.200.3` IP address, then the configuration will look as follows:

```
stream example {
  input udp://239.0.0.1:1234/10.100.200.3;
}
```

MPTS ingest

To ingest a multiprogram transport stream (MPTS), use the protocol-specific options instead of `udp://`.

Operating system tuning

Linux default settings do not allow ingesting video via UDP without loss, so you have to significantly increase the size of network buffers.

See detailed instructions on tuning the Linux network subsystem in Performance.

Note that to ingest HD video the recommended buffers size is about 16MB.

Multicast ingest issues

If you have any problems with the quality of ingested video, you should try to find what the problem is.

First of all, remove all iptables rules: `iptables -F`.

The rp_filter should also be disabled to avoid routing problems. *Flussonic* disables it automatically only on the network interface where you enable multicast ingest to reduce server sensitivity to network attacks. If, for some reason, rp_filter was not disabled by *Flussonic*, disable it manually:

```
sysctl -w 'net.ipv4.conf.eth0.rp_filter=0'
```

and

```
sysctl -w 'net.ipv4.conf.all.rp_filter=0'
```

Change `eth0` to real interface if it differs.

Second, note that when you watch video with Flussonic, many factors affect its quality: the signal quality, ingest quality, the server preformance, and the performance of your network. So the problem probably is not caused by Flussonic Media Server. Now let's try to find the source of problems.

If you run:

```
/opt/flussonic/contrib/multicast_capture.erl udp://239.0.0.1:1234/10.100.200.3 output.ts
```

and record 30 seconds of video, copy it to your computer and watch that video in VLC, then you will get an actual quality of multicast received by the server. This script does not extract the MPEG-TS but writes raw multicast to disk.

If at this stage you got a nice smooth video, you can go ahead and run this command on the server itself:

```
curl -o output.ts http://127.0.0.1:80/example/mpegts
```

You will receive the video that was ingested by Flussonic, unpacked and packed back in MPEG-TS. Download this file to your computer and watch it locally to make sure that the quality of your network connection does not affect the experiments.

If at this stage the video is also good, but when viewing from Flussonic it freezes, the problem most likely is that your network connection bandwidth is not enough to transfer video smoothly from Flussonic to you.

Issues with switches

Sometimes the settings of a network switch can cause issues. For example, one client had a problem with the limit on the number of received channels. It turned out that there was a limit on the number of subscriptions on one port. You can check this limit with the command:

```
#debug igmp snooping all
```

If you see this message:

```
%Jun 25 15:12:18 2015 SrcIP is 192.168.121.2, DstIP is 226.2.1.16
%Jun 25 15:12:18 2015 Groups joined have reached the limit, failed to add more groups
```

You can fix the problem by raising the limit:

```
#ip igmp snooping vlan XX limit group <1-65535>
```

Issues with multicast addresses

In certain cases, headends have issues with multicast group addresses between `224.0.0.0.0` and `239.1.1.1.1`. We recommend using multicast group addresses from `239.1.1.1.1` and higher. Lower values may not work.

3.2.12 Ingesting RTMP streams

Flussonic supports ingesting RTMP streams. Read more about RTMP at Using RTMP protocol.

The general RTMP ingest URL is:

`rtmp://FLUSSONIC-IP/application/stream_name`

The RTMP protocol requires that an RTMP URL has at least two segments. The first segment (`application`) is by default used as the name of the RTMP application. Read more about the features of URL when using the RTMP protocol here.

If the name of the RTMP application on the server consists of more than one segments, add two slashes to the URL in order to explicitly divide the RTMP application and stream name.

3.2.13 Ingesting SRT streams

Flussonic supports ingesting SRT streams. Read more about SRT at Using SRT protocol.

SRT INGEST URL

For *Flussonic* to ingest an SRT stream, you should create a stream with the input URL formatted according to one of the options below:

- Ingest using IP:PORT:

  `srt://SRT-SOURCE:SRT_PORT`

- SRT parameters in the URL parameters:

  `srt://SRT-SOURCE:SRT_PORT streamid="#!::m=request,r=STREAM_NAME"`

- SRT parameters in the URL query string:

  `srt://SRT-SOURCE:SRT_PORT?streamid=#!::m=request,r=STREAM_NAME`

where:

- `SRT-SOURCE` is an IP address of an SRT source server.
- `SRT_PORT` is an SRT port of an SRT source server.
- `streamid` is a string formatted as described here.
- `r=STREAM_NAME` is the name of a stream to ingest.

For example:

```
stream ingest_srt {
  input srt://SRT-SOURCE:8888 streamid="#!::m=request,r=srt_stream";
}
```

In the example above we enabled an ingest of an `srt_stream` stream over port `8888`.

PARAMETERS FOR AN SRT INGEST

Flussonic allows you to manage the ingest of the SRT streams by setting parameters.

Example with `passphrase`:

```
stream ingest_srt {
  input srt://SRT-SOURCE:9999 passphrase=0987654321 streamid="#!::m=request";
}
```

or:

```
stream ingest_srt {
  input srt://SRT-SOURCE:9999?streamid=#!::m=request&passphrase=0987654321;
}
```

In the examples above we secured stream port `9999` with `passphrase`.

Besides, here you can find a few parameters to specify in `streamid` as well:

- `u=USERNAME` is a username. *Flussonic* uses it as a token for an ingest session authorization.
- `a=USER_AGENT` is a user agent. Used in the ingest session authorization.

3.2.14 Accept publication over SRT

You can easily publish a stream via SRT to any server with Flussonic Media Server.

> **Note**
>
> If you have many servers and the number of active publishing sessions is unknown, use a more complex solution: SRT publishing from many authors.

Configuration in the UI

To configure publishing SRT streams to *Flussonic Media Server*:

1. Create a stream with empty **Source URL**, then set **Publication** to **Enabled** in the profile of that stream.
2. Specify **SRT (dedicated publish port)** and **passphrase** for publishing to this stream.
3. Save the settings. After that, a URL will be shown in **SRT (DEDICATED PUBLISH PORT)**. Use it for publishing.

   .

> **Note**
>
> The above instruction is for setting a separate port for publishing via SRT to a specific stream. If you want to use the global port for publishing via SRT to all streams, use the **SRT (SHARED)** link and set the port on the **Config** tab. Learn more about ports for SRT at SRT ports.

Testing the publication

Publish a stream to Flussonic, for example like this:

```
ffmpeg -re -i /opt/flussonic/priv/bunny.mp4 -c copy -y -f mpegts 'srt://localhost:9050?passphrase=mytopsecret'
```

Settings in the config file

You can configure the stream for publishing in the config file. Depending on way to specify SRT port, the settings are different:

- Global port (**SRT (SHARED)** in the UI)

```
srt_publish {
  port 9998;
  passphrase 0123456789;
}
stream pub {
  input publish://;
}
```

Use the URL for publishing:

`srt://FLUSSONIC-IP:SRT_PORT?passphrase=PASSWORD&streamid=#!::r=STREAM_NAME,m=publish`

- Separate global option for publishing

```
srt_publish {
  port 9998;
}
stream mysrt {
  input publish://;
}
```

Use the following link to publish the stream:

`srt://FLUSSONIC-IP:SRT_PORT?streamid=#!::r=STREAM_NAME`

- Separate port for a stream

You can allow publishing **and** playing a stream or a group of streams over single port. Specify the port in the `srt PORT_NUMBER` parameter in a stream or template settings:

```
stream mysrt {
  input publish://;
  srt 9998;
}
```

To publish the stream use the following link:

`srt://FLUSSONIC-IP:SRT_PORT?streamid=#!::m=publish`

, where:

- `FLUSSONIC-IP` is an IP address of your *Flussonic* server.
- `SRT_PORT` is an SRT port.
- `m=publish` is a publishing mode.
- Separate port for publishing a stream

```
stream pub {
  input publish://;
  srt_publish {
    port 9998;
    passphrase 0123456789;
  }
}
```

URL for publishing:

`srt://FLUSSONIC-IP:SRT_PORT?passphrase=PASSWORD`

Additional parameters for SRT publishing

The list of parameters you can set for `srt_publish` aside from `port` to manage SRT publication is given here.

Flussonic also carries such information as agent (*Flussonic* version) and session ID in the URL. You do not have to specify them manually.

3.2.15 WebRTC publishing

When you develop your own application or website for exchanging video and/or audio via WebRTC, you may choose to send streams to *Flussonic* to enrich your project with useful cutting edge features extending the basic protocol. You may find recommendations on embedding Flussonic's WebRTC capabilities to your app here. Below are the instructions on configuring *Flussonic* to accept WebRTC publications.

The publication is carried out via the WHIP standard. For more details about WebRTC, WHIP and WHEP, see Using WebRTC protocol.

On this page:

- WebRTC stream settings in Flussonic.
- WebRTC publication options.

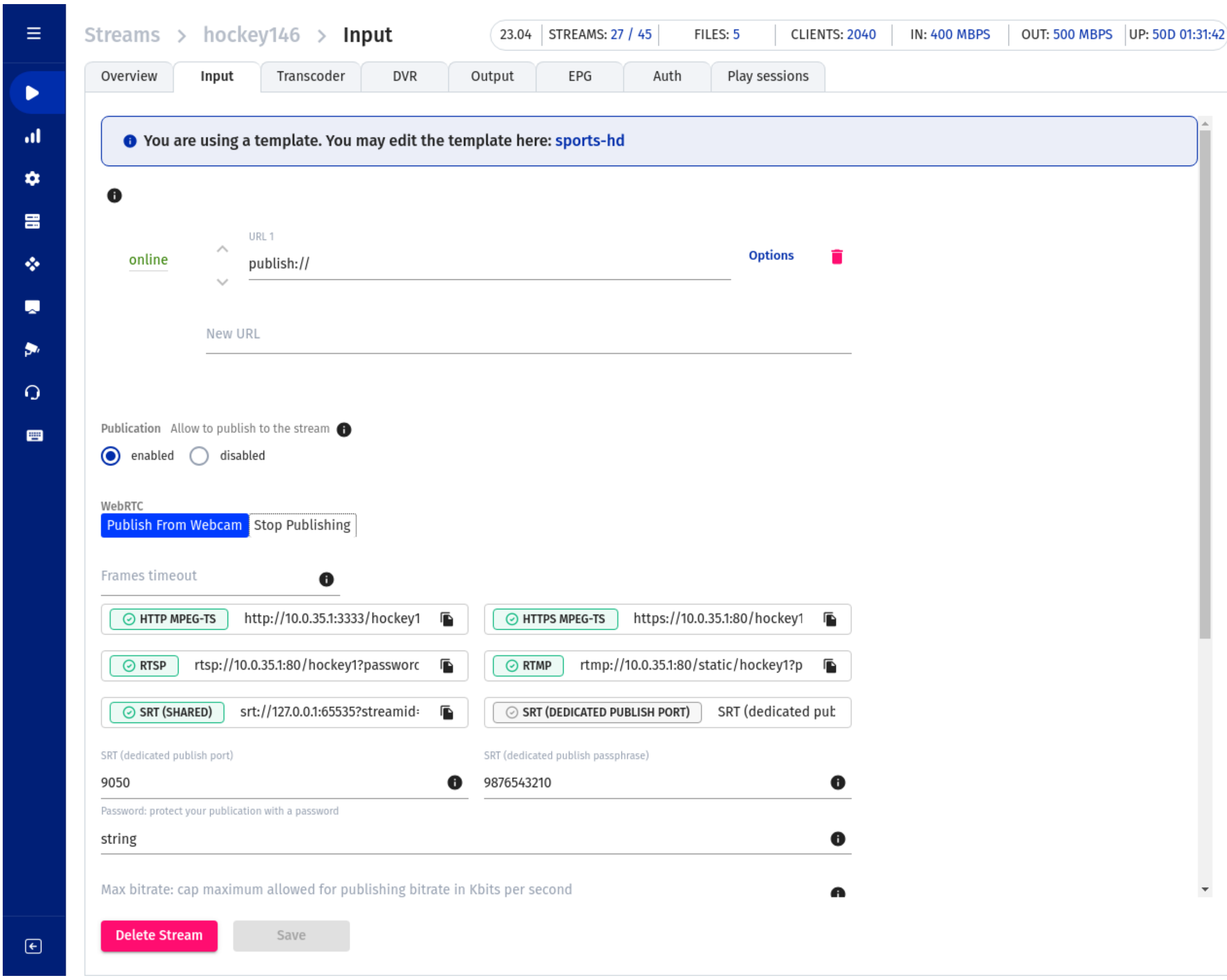
Prerequisites

- Configure HTTPS. Most of the modern browsers allow video and audio publishing via WebRTC over a secure connection only.
- Do not close UDP ports. *Flussonic* does not have a fixed port range for WebRTC, i.e. the ports allocated by the OS are used. When closing UDP ports, you can accidentally close the ones that are being used. Consider not using any other software on the *Flussonic* server to safely allow UDP listening on all ports.

WebRTC publication settings in Flussonic

To accept WebRTC publication on the *Flussonic* server, just create a stream with `publish://` input in one of the following ways:

- In the UI as described here
- Via the API: `Flussonic-API: PUT /streamer/api/v3/streams/{name}`



Use the following URL for the published stream in your client application:

```
http://FLUSSONIC-IP:PORT/STREAM_NAME/whip
```

See also Streaming API reference to learn how to compile the payload for this URL.

Optional settings

Having created a stream with `publish://` input, you can start publishing to it right away. No additional settings are strictly necessary, but you may need them to achieve certain objectives. Below are possible further settings of WebRTC publishing.

STREAM PARAMETERS

You can configure transcoding of the WebRTC published streams at **options** on the same **Input** tab. The following settings are available in the UI:

- Output audio codec is an audio transcoding option.
- Maximum and minimum bitrate are the basic settings for Adaptive publication over WebRTC.

> **Note**
>
> If you want to publish with a dynamic name, these settings are set in the template instead of the stream. The full list of WebRTC stream options is available in the API schema.

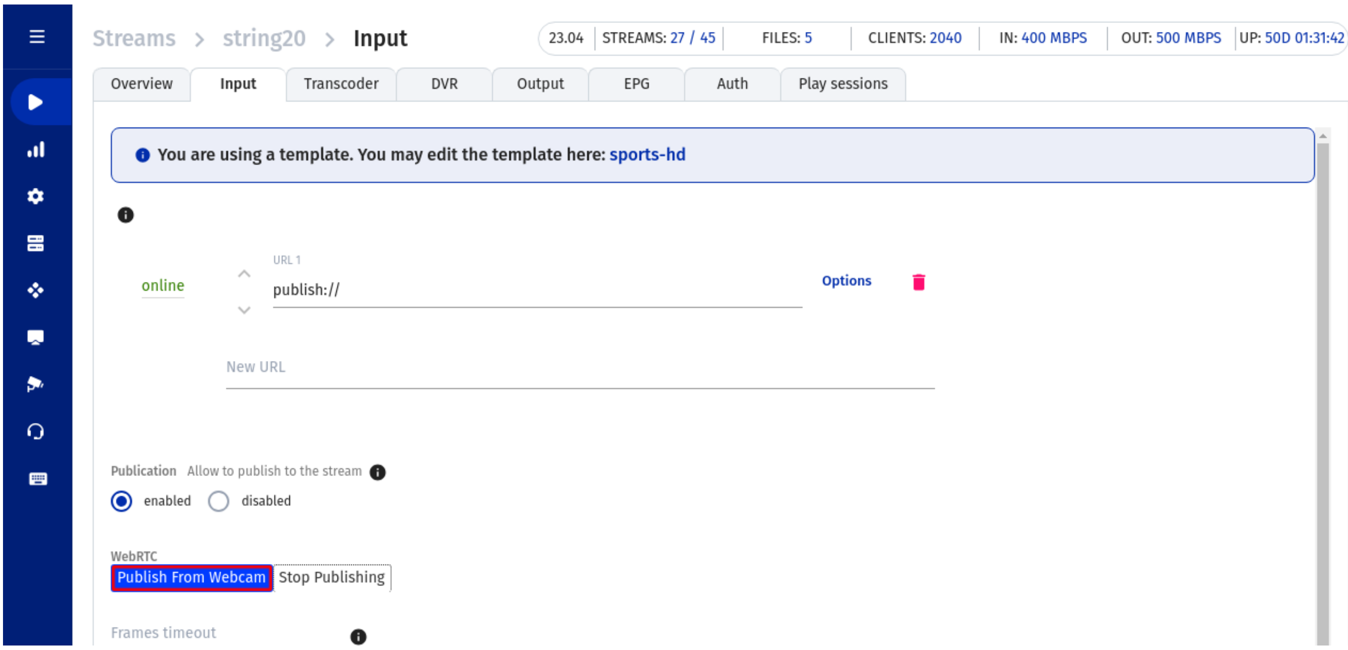
PROTECTING YOUR PUBLICATION

To prevent anyone other than your users from publishing videos to the server, you can protect the publication in one of the following ways:

- Protect the publication with a password.
- Configure external authorization for the publication sessions.

TEST PUBLISHING FROM YOUR WEBCAM

To make sure the settings are correct, you can try publishing from your webcam to the created stream. Just click **Publish From Webcam** on the **Input** tab in the stream profile and allow your browser to access your camera and microphone.



The webcam video preview window will be displayed under the button.

> **Warning**
>
> This publishing method is available for testing purposes only. End users usually do not have access to the *Flussonic* admin UI; they can publish videos through your application (website).

PUBLISHING WITH ADAPTIVE BITRATE

Publishing to *Flussonic* is performed at Adaptive Bit Rate (ABR) by default. This means that *Flussonic* helps the publication source adjust the bitrate to the bandwidth. Learn more at Adaptive publication over WebRTC.

LOAD BALANCING WITH WHIP PUBLICATIONS

Since WHIP is based on HTTP POST requests, you can use our load balancer to distribute play requests between servers in a cluster. The balancer will redirect POST requests to servers in the cluster using the 307 HTTP redirect code.

WEBRTC PLAYER'S PUBLICATION OPTIONS

If you use our *WebRTC Player* following the recommendations on developing the client application, you can flexibly configure the publications. You will find detailed instructions and examples in the readme. For example, you can configure it to perform:

- Audio podcasts through WebRTC. Just set `video: false` and `audio: true` in the Publisher's `constraints`.
- Custom filters application via `canvas`. The example of a webpage incorporating *WebRTC Player* also includes the use of `canvas`. If you try the example, you will see that "sepia" filter is applied to the published video and "It's publishing to Flussonic!" text appears over it.

3.2.16 Publishing RTMP streams

Flussonic supports publishing RTMP streams. Publishing streams via the RTMP protocol is widely used when delivering live video over the Internet, because RTMP guarantees content delivery and allows for low latency. Read more about RTMP at Using RTMP protocol.

The publication is configured in a usual way as described at Publishing Video to the Server. Please refer to The Publication from OBS Studio to Flussonic Media Server for the example of configuration.

Below are URLs that you can use to publish from a third party software to *Flussonic*. Read more about the features of URL when using the RTMP protocol here.

> **Note**
>
> Make sure to set ports in order to publish via RTMP or RTMPS to *Flussonic*.

Configuration procedure

To configure publishing SRT streams to *Flussonic Media Server*:

1. Create a stream with `input publish://` as described here.
2. Make sure to setup an RTMP port.
3. Publish a stream to Flussonic, for example as shown here.

Publishing to a static stream

You can use the following RTMP URLs for publishing to a static stream:

- `rtmp://FLUSSONIC-IP/stream_name`. In this case, the application name should be specified as part of the stream name in Flussonic. For example, if you use stream name `client15/published1`, then specify in your third party software:
  - server URL: `rtmp://FLUSSONIC-IP/client15`
  - stream name: `published1`
- `rtmp://FLUSSONIC-IP/static/stream_name`. In this case, in a third-party application you can specify the application name `static`, and *Flussonic* will recognize this reserved word as the application name and discard it:
  - server URL: `rtmp://FLUSSONIC-IP/static`
  - stream name: `published`

  > **Note**
  >
  > However, if you have explicitly configured a stream with complex `static/published` name in Flussonic, the `static` part will be considered as a part of the stream name.

Publishing to a dynamic stream

Flussonic uses the following logic when publishing via RTMP to a dynamic stream:

1. The server concatenates the application name with the path being published. Thus, the pairs `rtmp://FLUSSONIC-IP/chats/my`, `chat-15` and `rtmp://FLUSSONIC-IP/chats`, `my/chat-15` produce the published stream name `chats/my/chat-15`
2. The program searches for the first publishing prefix this name contains. In the above examples, that would be the prefix named `template`.
3. Then, all authorization interfaces and the like use the complete stream name: `chats/my/chat-15`.

Please note that you should not use the the reserved `static` part in the URL for RTMP publishing with a dynamic name. If you do so, the `static` part will be considered as an application name and omitted when creating a stream in *Flussonic*. For example:

`rtmp://flussonic-ip/static/test`

In this case, a stream `test` will be created in *Flussonic*. If you don't want `static` to be omitted, configure explicitly a stream for publishing with the static name `static/test` instead of using a template.

Transcoding audio

If published RTMP streams contain audio in PCMU, then you can transcode it to AAC, or specify that the audio tracks must not be transcoded:

- `output_audio=(keep|add_aac|aac)` . Specifies audio transcoding options. You can get the resulting audio for playback in AAC (`aac`), AAC+PCMU (`add_aac`) or leave the original codec without change (`keep`). By default, `keep` is used.

```
template chats {
  prefix chats;
  input publish://;
  output_audio aac;
}
```

3.2.17 Publishing from OBS Studio to Flussonic Media Server

Publishing from OBS Studio to Flussonic Media Server

Using Open Broadcaster Software (OBS) you can publish a stream from your computer to Flussonic Media Server. This can be used for streaming games, webinars and any other broadcasts from a computer to the Internet.

For example, you can stream to social networks.

Content:

Publishing to a static stream in Flussonic Media Server

SETTING UP PUBLISHING VIA THE CONFIGURATION FILE

In *Flussonic Media Server*, it's enough to create a stream and indicate that you allow the publication in it.

In the configuration file `/etc/flussonic/flussonic.conf` add the stream:

```
stream published {
  input publish://;
}
```

To apply the settings, run:

```
service flussonic reload
```

Read more in Publishing video to the server to a static stream.

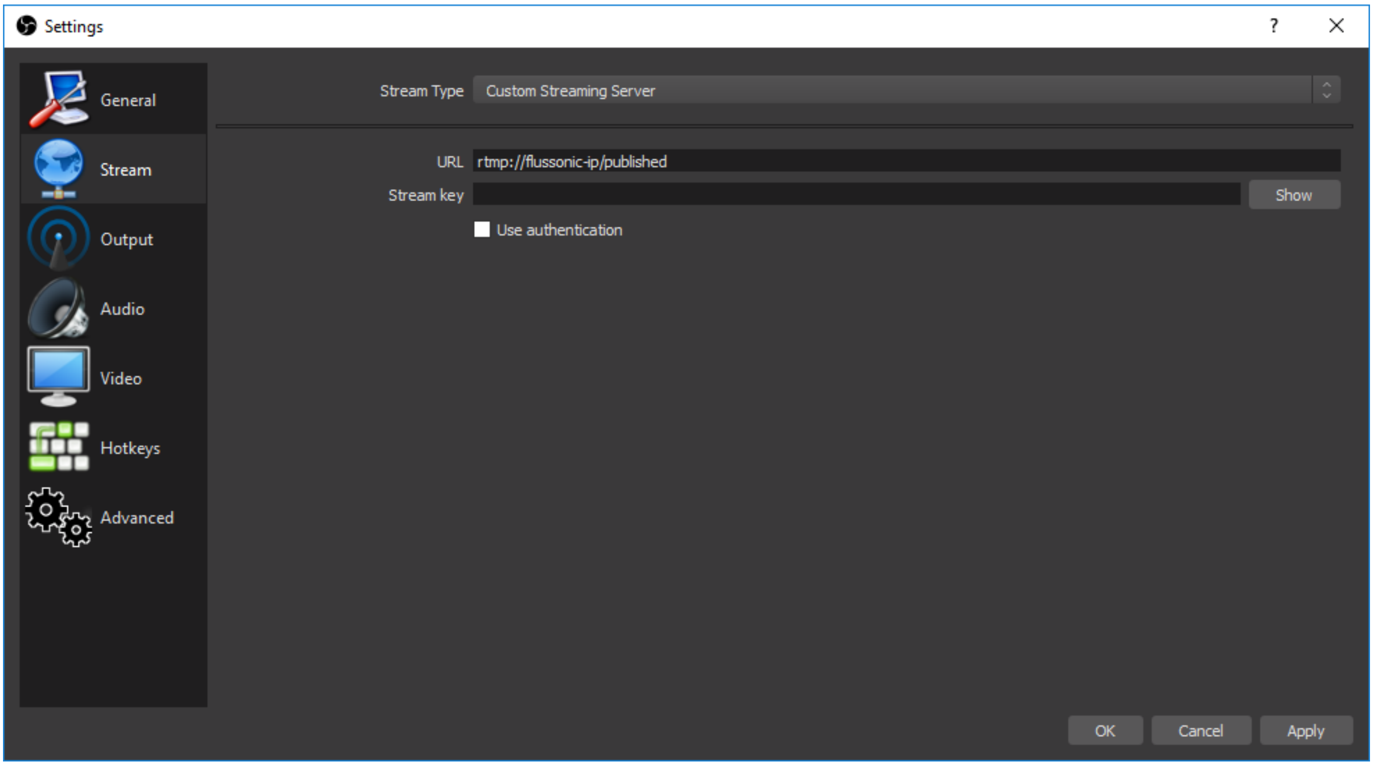
Static streaming from OBS Studio

Download and install OBS Studio. Open the program and go to settings.

Open the menu **Stream**:

- **Stream Type**: Custom Streaming Server
- **URL**: `rtmp://flussonic-ip/published`
- **Stream Key**: leave empty

Where `published` is the name of your stream.



Click **OK** to save.

Publishing under a dynamic name in Flussonic Media Server

If a name of a publishing stream is not known in advance or you expect multiple streams, you can specify the *publication prefix*.

For more information, see: Publishing with a dynamic name.

SETTING UP PUBLISHING VIA THE CONFIGURATION FILE

In the configuration file `/etc/flussonic/flussonic.conf` add the stream:

```
template chats {
  prefix chats;
  input publish://;
}
```

To apply the settings, run:

```
service flussonic reload
```

Read more in Publishing with a dynamic name.

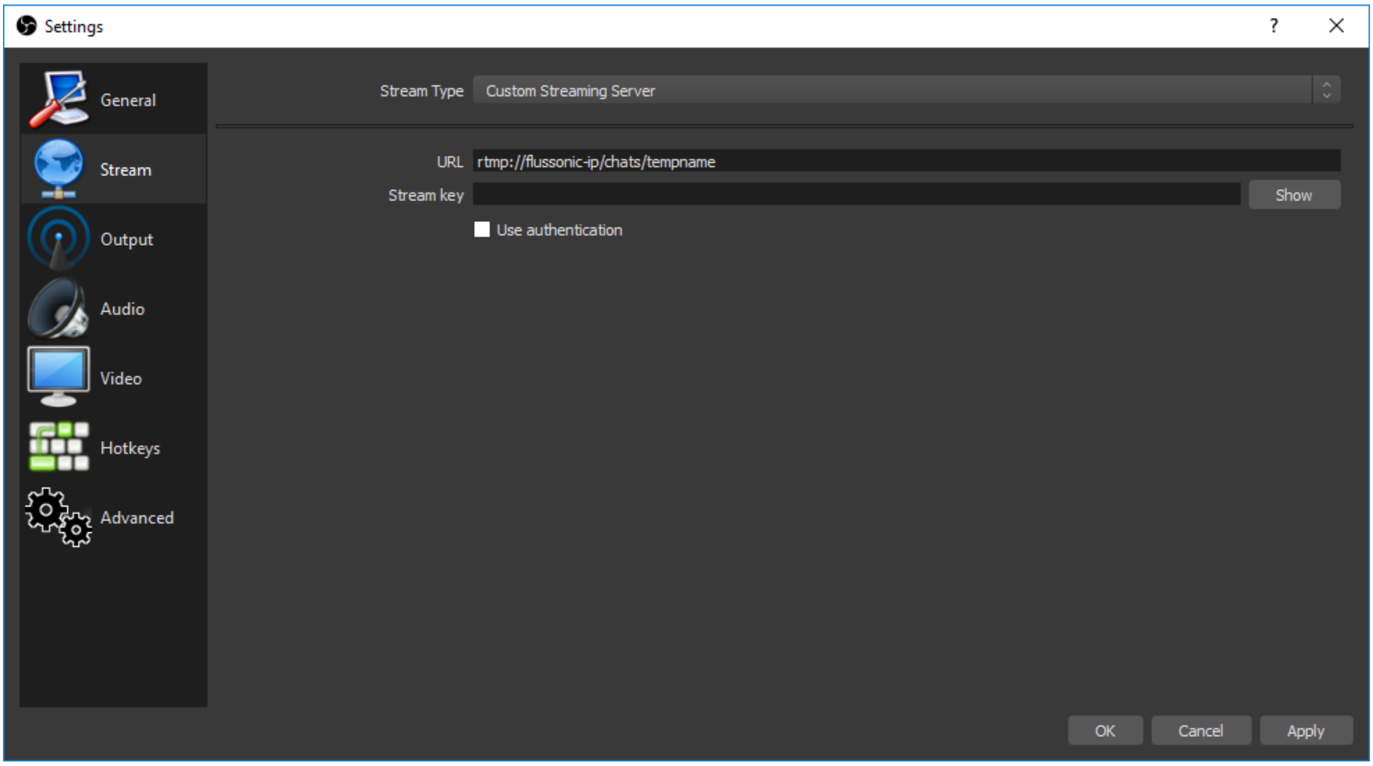
Broadcast dynamic stream from OBS Studio

Download and install OBS Studio. Open the program and go to settings.

Open the menu **Stream**:

- **Stream Type**: Custom Streaming Server
- **URL**: `rtmp://flussonic-ip/chats/tempname`
- **Stream Key**: leave empty

Where `chats` is the name of the prefix. What comes after `chats` depends on the client. Flussonic Media Server does not know in advance what it will be.



Click **OK** to save the changes.

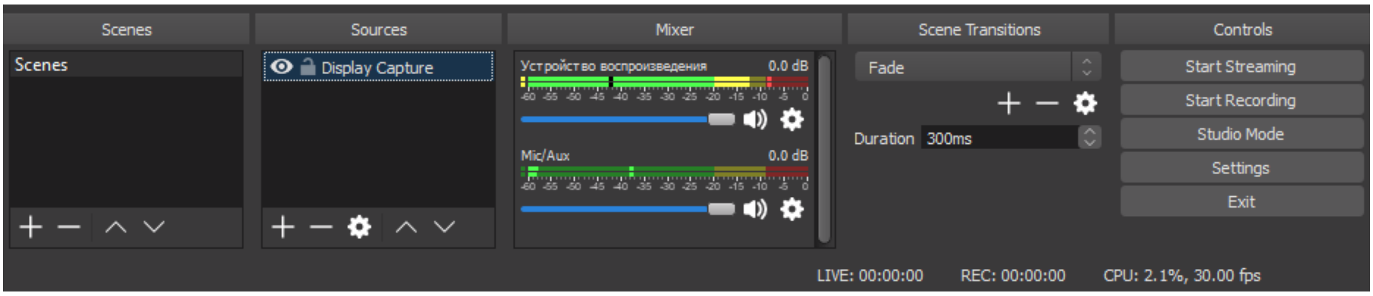
In the main OBS Studio window, click **Start Streaming**.

The broadcast has already started and you can watch it in the administrative interface of Flussonic Media Server. For now it's just a black screen. So stop the streaming and configure OBS Server.

Configuring OBS Studio

Open the OBS Studio's main window and create a scene. For example, "Stub", "Live", "Break", "End".

> **Warning**
>
> All the scenes and sources in OBS Studio are common and cannot have the same name. If you called a source "Live Broadcast", then you cannot call a scene the same name.

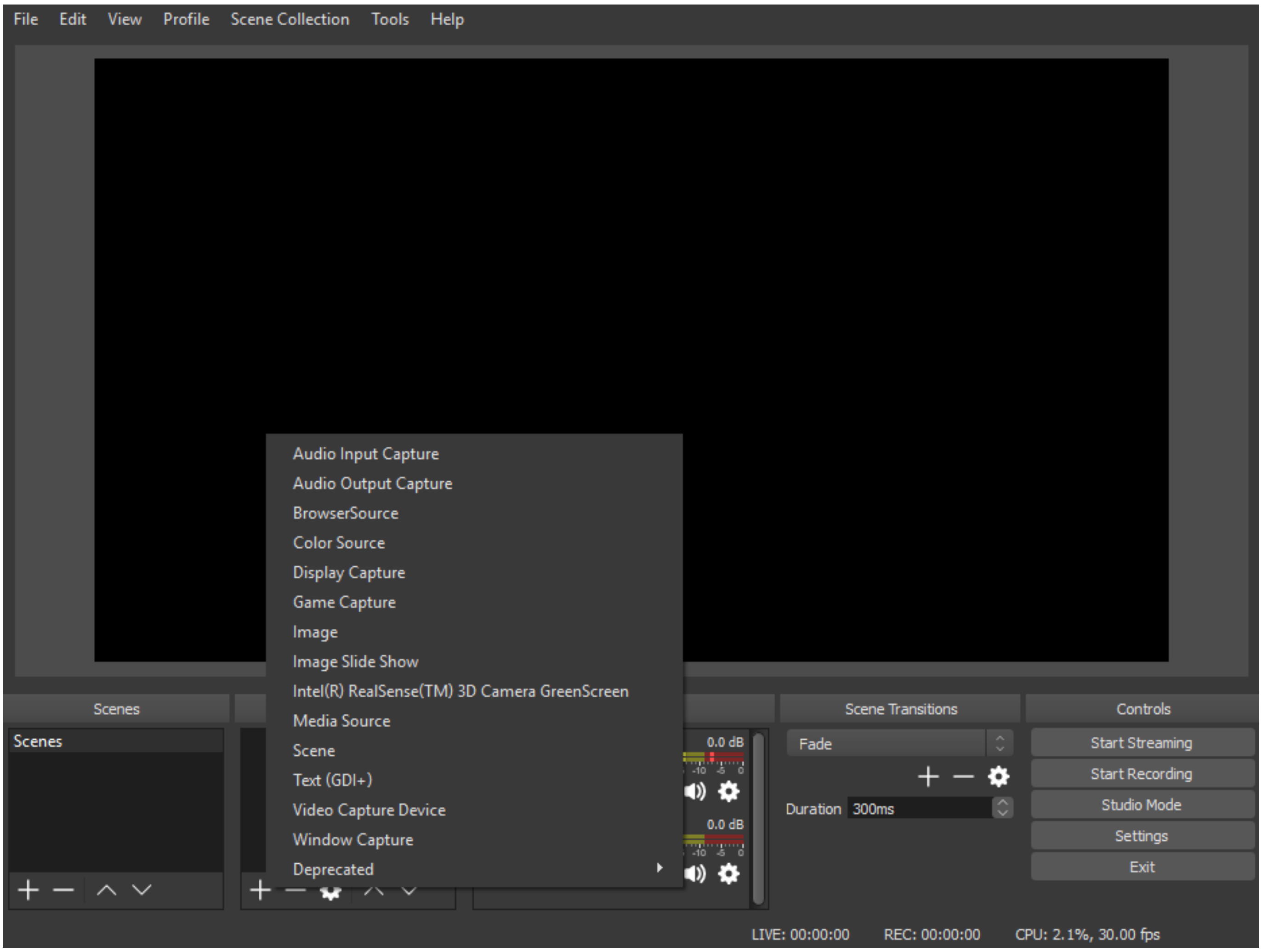


In each scene you can add different sources of broadcast. A source can be a whole screen or a separate open window. For example, a running application, a browser or even a separate browser tab. At any place on the screen, you can display text, a media source or a stream from a webcam.

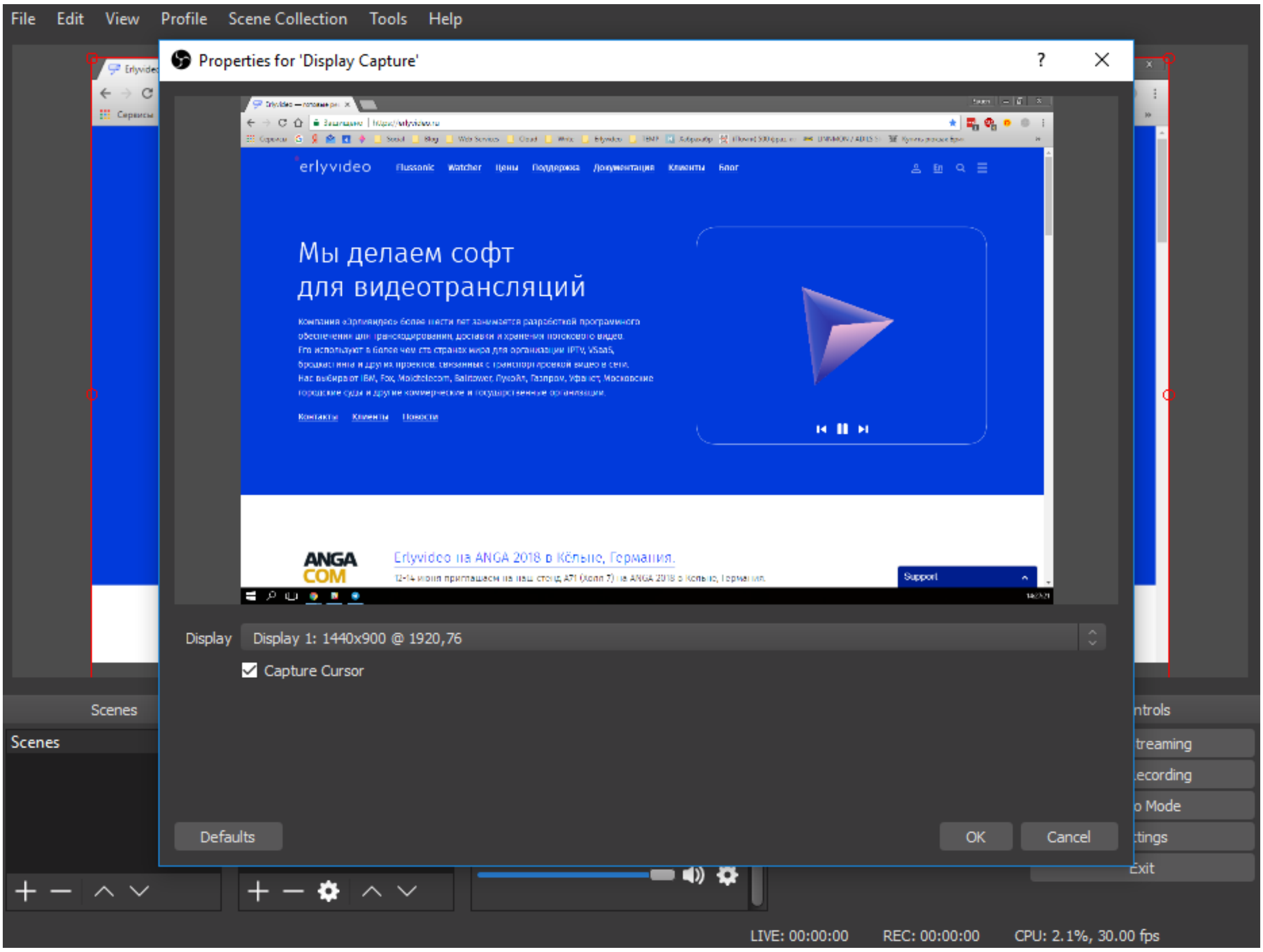
You can change the order of the sources by dragging them along the list or using the up and down arrow buttons.

The source that is located higher in the list will be a priority and will "hide" the ones below it.

You can turn on and off the sources during the broadcast.



Example of setting display capture:



When the source is selected in the list, you see a red frame around it.

This is a bounding box that can be used to move sources when previewing.

You can enlarge or reduce the frame.

Hotkeys that are available when previewing to change the position and size of the source:

- Press the "Ctrl", to disable the binding source/border.
- Press the "Alt" and drag the bounding box to trim.
- "Ctrl + Alt" — fit to fit the screen.
- "Ctrl + S" — stretch to full screen.
- "Ctrl + D" — to place on the center of the screen.
- "Ctrl + R" — size/source position reset.

In the menu **Mixer**, you can adjust the volume of the connected audio channels.

In the menu **Scene Transitions**, you can choose how switching between scenes will work: by fade or cut (immediate switching).

3.2.18 H323

Flussonic Media Server can call via VoIP protocol H323 (for example, to Polycom devices) and ingest video data via H323.

Configuration example:

```
stream polycom1 {
  input h323://192.168.100.150 vb=2000k id="Flussonic";
}
```

Flussonic will connect to the specified hostname and will encode video with **2000k** bitrate. Audio will automatically be transcoded to AAC.

With the **id** option you can specify the name that will be displayed displayed on a remote H323 device when Flussonic connects to it.

3.2.19 Adaptive publishing over WebRTC

When a client device is publishing from a browser to *Flussonic*, *Flussonic* controls the browser from which the publication is carried out so that the browser adjusts the bitrate of the publication to the bandwidth of the channel. This prevents packet loss when the bandwidth of the Internet connection is insufficient. If you reduce the channel width, the client must reduce the publication bitrate, if you expand the channel, the client must increase the publication bitrate.

This feature is enabled by default and usually does not require any configuration because the default settings are optimal. However, you can change the values as you wish to achieve even more performance.

ABR publication settings in the Flussonic UI

To specify additional options for a published source, click **options** next to the source URL. Adaptive bitrate settings are under **WebRTC** > **ABR**:

The following settings in the file correspond to these settings in the UI:

```
stream published_stream_name {
  input publish:// abr_loss_lower=2 abr_loss_upper=10 abr_mode=1 abr_stepdown=50 frames_timeout=1 abr_max_bitrate=2200 min_bitrate=500 output_audio=aac
priority=0 source_timeout=5;
}
```

Flussonic recommends the browser bitrate within the `min_bitrate - abr_max_bitrate` range, depending on the presence and amount of packet losses during publication.
Flussonic recommends lowering the bitrate when the amount of losses is **more** than `abr_loss_upper` and increasing it when the amount of losses is **less** than `abr_loss_lower`. Decreasing and increasing are performed by steps of size `abr_stepdown` and `abr_stepup`, respectively. After the specified number of auto-adjustment cycles (`abr_cycles`) passes, *Flussonic* considers the bitrate to be optimal, and it is no longer analyzed. By default, `abr_cycles=5`. If `abr_cycles=0`, the adjustment process takes place all the time while the publication lasts.

Also, *Flussonic* calculates the actual maximum bitrate. It remembers the bitrate values at which the losses grew up to `abr_loss_upper` and considers their average value over the past number of cycles to be the new maximum bitrate value (current).

3.2.20 SRT publishing from many authors

To accept SRT publishing from multiple authors while centrally issuing a unique passphrase to each author, use the `srt_port_resolve` mechanism offered by Flussonic for SRT publishing to the cloud. For more information about this solution, see SRT page.

> **Note**
>
> An alternative is to add streams to the Flussonic configuration manually. But this method doesn't work well when you have dozens of servers and thousands of authors: you will need to somehow determine which server to receive a stream from this or that author, and the passphrase will be stored in configuration files that are not automatically synchronized on the servers.

Step 1. Develop a configuration backend

Each author publishes their content to a pre-allocated port with own passphrase. A configuration backend manages the stream configuration to match ports to stream names and stream names to passphrases. Below is a simple example of such a configuration backend.

```
from http.server import BaseHTTPRequestHandler, HTTPServer
import urllib.parse as urlparse

class RequestHandler(BaseHTTPRequestHandler):
    streams_config = {
        "streams": [
            {
                "name": "publish_stream_name",
                "inputs": [
                    {"url": "publish://"}
                ],
                "srt_publish": {"passphrase": "securePass"}
            },
            {
                "name": "other_publish_stream",
                "inputs": [
                    {"url": "publish://"}
                ],
                "srt_publish": {"passphrase": "otherPassword"}
            }
        ]
    }

    port_stream_mapping = (
        (2345, "publish_stream_name"),
        (2346, "other_publish_stream"),
    )

    def do_GET(self):
        parsed_path = urlparse.urlparse(self.path)
        path = parsed_path.path
        query_components = urlparse.parse_qs(parsed_path.query)

        # Handle /config_backend/streams
        if path.startswith('/config_backend/streams'):
            self.send_response(200)
            self.send_header('Content-Type', 'application/json')
            self.end_headers()
            response = self.streams_config
            self.wfile.write(bytes(str(response), "utf8"))

        # Handle /config_backend/srt_port_resolve/{port}
        elif path.startswith('/config_backend/srt_port_resolve/'):
            port = int(path.split('/')[-1])
            mode = query_components.get('mode', [None])[0]
            host = query_components.get('host', [None])[0]

            response = next((stream_name for port_number, stream_name in self.port_stream_mapping if port_number == port), None)

            if response and mode == 'publish':
                self.send_response(200)
                self.send_header('Content-Type', 'text/plain')
                self.end_headers()
                self.wfile.write(bytes(response, "utf8"))
            else:
                self.send_error(400, "Invalid port or mode.")
                return

        else:
            self.send_error(404, "404 Not Found")

def run(server_class=HTTPServer, handler_class=RequestHandler, port=12345):
    server_address = ('', port)
    httpd = server_class(server_address, handler_class)
```

```
    print(f'Starting httpd on port {port}...')
    httpd.serve_forever()

if __name__ == "__main__":
    from sys import argv

    if len(argv) == 2:
        run(port=int(argv[1]))
    else:
        run()
```

To run this configuration backend:

1. Save the above code to the file `/tmp/config_server.py` using any text editor.
2. Install Python if not already installed.

   `apt install python3`

3. Launch the configuration backend on a free port, for example, 12345.

   `python3 /tmp/server.py 12345`

Step 2. Configure Flussonic to use the configuration backend

You have to tell *Flussonic* where to fetch streams and which ports to listen for SRT publications. Send the configuration backend address and the SRT port range via PUT /streamer/api/v3/config API call:

```
curl -u USER:PASSWORD -X PUT http://localhost:80/streamer/api/v3/config \
-H "Content-Type: application/json" \
--data '{"listeners": {"srt": [ {"mode":"publish","ports":{"first":2345,"last":12344}}]},"config_external": "http://localhost:12345/config_backend/"}'
```

The `publish_stream_name` and `other_publish_stream` streams retrieved by *Flussonic* from the configuration backend should be created in the UI within 5-10 seconds after executing this command.

Step 3. Test the setup

Publish the stream to *Flussonic*:

```
ffmpeg -re -i /opt/flussonic/priv/bunny.mp4 -c copy -y -f mpegts 'srt://localhost:2345?passphrase=securePass'
```

Check that the video is playing in the `publish_stream_name` stream profile on the **Overview** tab in the UI.

Related objectives

- Centralized cluster configuration with config_external

3.3 Process

3.3.1 Mixer

Flussonic Media Server can create a new stream that uses other streams as its video and audio sources. This chapter demonstrates how to add a mixer stream that takes video from a surveillance camera and audio from an internet radio broadcast.

Adding a mixer stream

Create a new stream and specify the `mixer://` schema and two streams as its source. The first stream will provide video, the second one — audio:

```
stream mix {
    input mixer://stream1,stream2;
}
```

Here:

- `stream1` is the name of the live stream that will provide the video track.
- `stream2` is the name of the live stream that will provide the audio track.

> **Warning**
>
> The mixer works only with streams that have already been added to Flussonic Media Server. The only data sources you can use in the `mixer://` directive are streams, and not VOD files or data source URLs.

Usage example

Imagine you have a video stream from a surveillance camera which has been installed on a tall pole. The audio isn't useful, because the only thing you can hear is the wind.

You might wish to disable the sound coming from the source:

```
stream camera {
  input fake://fake;
}
stream silent {
  input rtsp://localhost/camera tracks=1;
}
```

Alternatively, you can create a new stream with video from the camera and audio from another source using the mixer.

```
stream origin {
  input fake://fake;
}
stream cam1 {
  input rtsp://localhost/origin tracks=1;
}
stream radio {
  input shout://localhost/origin/shoutcast;
}
stream cam1radio {
  input mixer://cam1,radio;
}
```

You have created a `cam1radio` stream, which replaces the audio coming from the camera with an internet radio stream. Viewers will be able to listen to the radio while watching the video stream, which may be useful during emergency situations.

You can also archive the original video and audio using the DVR feature:

```
stream cam1 {
  input rtsp://cam1.local/h264;
```

```
  dvr /storage 7d;
}
```

3.3.2 Mosaic

Flussonic has a built-in mosaic module. This module allows you to merge several streams into one view (a mosaic) and play it back as if it was a single stream. Mosaics are created with the use of the transcoder.

Merging streams into a mosaic

In the Watcher's web interface you can create client-side mosaic that shows several cameras at once. Learn more in Watcher documentation.

To create a server-side mosaic:

Install the `flussonic-transcoder` package:

Note. The package `flussonic-transcoder` is necessary only if you plan to use the CPU to perform transcoding. If you use Nvidia NVENC, no extra packages are needed.

```
apt-get -y install flussonic-transcoder
```

Now specify the following in the Flussonic configuration file:

```
stream cam1 {
  input rtsp://IP-CAMERA-ADDRESS:PORT/camera1;
}
stream cam2 {
  input rtsp://IP-CAMERA-ADDRESS:PORT/camera2;
}
stream cam3 {
  input rtsp://IP-CAMERA-ADDRESS:PORT/camera3;
}
stream cam4 {
  input rtsp://IP-CAMERA-ADDRESS:PORT/camera4;
}
stream mosaic0 {
  input mosaic://cam1,cam2,cam3,cam4?fps=20&preset=ultrafast&bitrate=1024k&size=340x240&mosaic_size=16;
}
```

After specifying the pseudo-URL `mosaic://` you need to type stream names separated by commas.

The option `fps=20` specifies frames per second for video. You can use `fps=video` for binding fps of mosaic to the first camera's stream.

The option `size=320x240` reduces the size of each stream in mosaic to the specified width and height.

The option `mosaic_size` tells how many slots should be in mosaic. Useful for specifying a fixed mosaic size.

3.3.3 Silence detection

Silence detection can be helpful for test purposes, for example, if you need to check your audio equipment for workabilty. For this, it would be useful to have an active working stream source and some indication when silence occurs in it.

Flussonic allows you to turn on silence detection on a stream and specify a threshold value of the sound level to tell Flussonic what it must consider silence. Flussonic will then generate events to inform you when silence occures and when the sound reappears. The events are generated only for active sources, not for lost ones. When a lost source reappears, Flussonic resumes to detect silence.

If a stream contains a number of audio tracks, Flussonic detects silence in the first of them.

To enable silence detection on a stream:

1. Open the Flussonic configuration file.
2. Add the `silencedetect` option into the stream configuration:

```
stream example {
  input udp://127.0.0.1:5500;
  silencedetect duration=20 interval=10 noise=-30dB;
}
```

   Here:

   - `duration` (in seconds) — the duration of a continuous time interval during which silence must last for Flussonic to generate an appropriate event.
   - `interval` (in seconds) — Flussonic will keep sending the event `audio_silence_detected` once upon the specified time interval until the sound reappears in the source.
   - `noise` — the threshold value of the sound level. Sound of this and lower level will be considered by Flussonic as silence.

   The configuration in the example means that if the sound is not louder than -30dB for at least 20 seconds, then Flussonic starts to generate the event `audio_silence_detected` every 10 seconds until the sound reappears.

3. Subscribe to the events `audio_silence_detected` and `audio_silence_end`, for example:

```
notify events {
  sink log:///var/log/flussonic/audio_silence.log;
  only event=stream_media_info,audio_silence_detected,audio_silence_end;
}
```

   Here:

   - `audio_silence_detected` — this event is generated when the sound level is not higher than the value specified in `noise`, for the time specified in `duration`.
   - `audio_silence_end` — this event is generated when the sound reappears in the source.

3.3.4 Copying streams

Flussonic allows you to copy a stream using the `copy://` source type. This function is needed in cases where it is impossible or too expensive to maintain multiple connections with the source of the video signal:

- For SDI capture cards. Flussonic connects directly to such cards and cannot establish more than one connection. If you have only one capture card and you need to perform a load test to see how the system behaves with a large number of streams, then you can use the `copy://` option to emulate more sources than you actually have.
- For RTSP sources like IP cameras. The connection with the RTSP source is a Unicast. If you need to receive a stream from a camera several times, then each copy will need a separate connection, and this increases the load on the network. Thanks to the `copy://` option, you can get the stream just once and "replicate" it on the Flussonic server.
- You can also copy only a specific set of tracks to a stream `copy://original?filter.tracks=v2a1`

With other source types, it is usually technically possible to receive several copies of the signal without special options, for example, the number of multicast or satellite signal receptions is not limited in any way.

Example of using the `copy://` option with a Decklink card to test the transcoder performance:

```
stream s {
  input decklink://0;
}

stream s1 {
  input copy://s;
  transcoder vb=1000k ab=64k external=false;
}
stream s2 {
  input copy://s;
  transcoder vb=1000k ab=64k external=false;
}
stream s3 {
  input copy://s;
  transcoder vb=1000k ab=64k external=false;
}
```

3.4 Transcode

3.4.1 Transcoder

Transcoding is essential if you want to:

- create a multi-bitrate stream
- change parameters of video — the codec and the bitrate of the stream, the frame size
- overlay a logo on a video stream

Transcoding implies two steps:

1. **Decoding**—a process of decompressing compressed data into a raw format.
2. **Encoding**—a process of compressing raw uncompressed data to be sent over the Internet. During encoding, various video parameters, such as resolution, bitrate, and type of compression, are determined.

Both encoder and decoder rely on **codec** — an algorithm of video and audio compression. A video codec affects file size and image quality. H.264 and AAC are the most commonly used video and audio codecs for live streaming.

> **Note**
>
> Remember that different protocols support different containers and codecs.

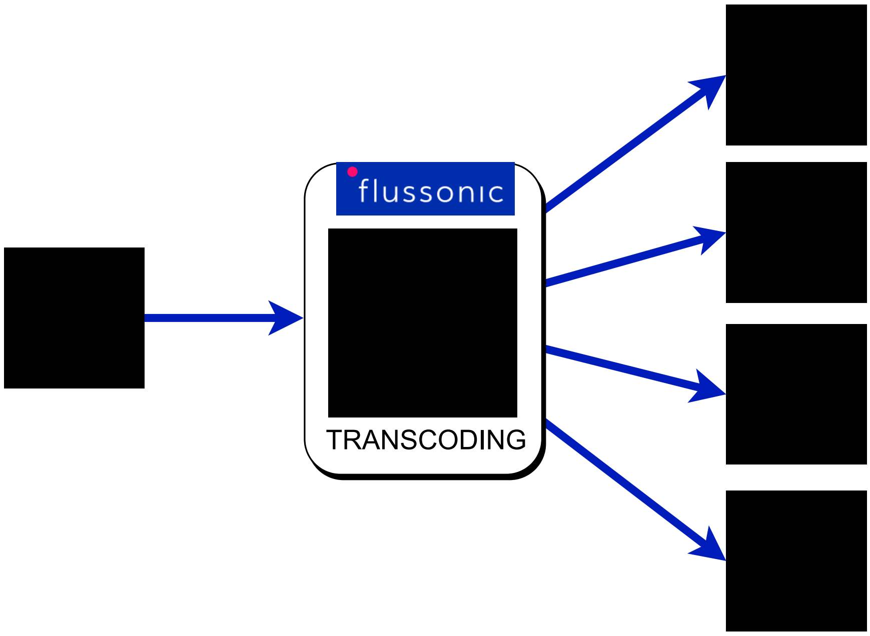
Transcoding is a process of decoding the stream, performing modifications to the video or audio parameters of the content, and then re-encoding the stream to be transmitted over the Internet.

Transcoding is commonly referred to as an umbrella term for one or a combination of the following media tasks:

- **Transcoding**: a process of changing a video or audio codec, or a type of compression, such as taking MPEG2 video and converting it to H.264 video.
- **Transsizing**: a process of resizing a video frame and modifying resolution, such as bringing down 3840×2160 (4K) resolution to 1920×1080p.
- **Transrating**: a process of lowering bitrates, without modifying codec or resolution, such as taking a 4K video at 45 Mbps and converting it to 4K at 15 Mbps.

The example below shows how the multi-bitrate stream is created when transcoding a 4K video at 45 Mbps:

Diagram 1. Example of transcoding



Flussonic Media Server has a built-in transcoder. It supports transcoding by using a GPU or CPU.

The transcoder module works with every input source supported by the *Flussonic Media Server*. The HLS protocol is partially supported — some sources might fail to be transcoded. It is recommended to test each HLS source manually to find out whether it works after transcoding.

For hardware-accelerated transcoding, *Flussonic* can use Intel and Nvidia solutions. When transcoding with Nvidia NVENC, streams with a 10-bit color depth are supported.

The transcoder seamlessly switches to another source without dropping frames. It is done by keeping the resolution of the original video in the resulting stream. The transcoder converts source video resolutions to be the same and match the value of the `size` parameter. If the `size` parameter is not specified, the output video stream resolution matches the resolution of the first video source received at the transcoder input.

Table of contents:

> **Note**
>
> Transcoding is an extremely computationally-intensive process, and it includes the following steps:
>
> 1. Decoding of the source stream into raw video data.
> 2. Processing and encoding of the raw stream according to the specified parameters.
>
> Depending on the configuration, a single server can process from 5 to 20 channels.

Installing the transcoder

If you use an NVENC-capable Nvidia GPU to perform transcoding, you do not need to install any extra packages to enable transcoding.

If you use the CPU to perform transcoding, you will need the `flussonic-transcoder` package installed. At default system settings, it is installed along with *Flussonic Media Server* as a recommended package. If you have disabled installation of recommended packages, install the transcoder package manually:

```
apt-get -y install flussonic-transcoder
```

This package can be found in the same repository as the `flussonic` package.

Configuring transcoder

Transcoder has plenty of options that can be divided into the following main groups:

1. Global options (applied to all video tracks)
2. Video encoding options (individual for each video track)
3. Audio encoding options (applied to all audio tracks)

All the options are described in details in the Transcoder configuration options section.

You can configure transcoder options by one of three ways:

1. Flussonic web-interface (recommended)
2. Configuration file
3. Flussonic API (see API reference)

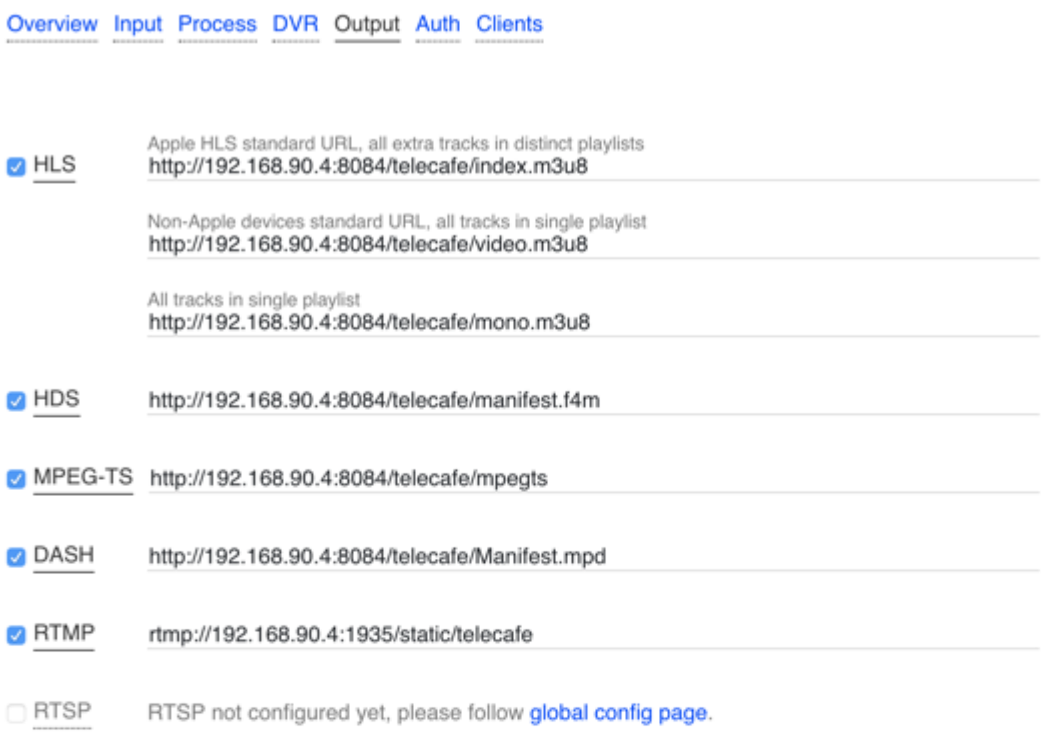
CONFIGURING TRANSCODER VIA THE WEB INTERFACE

Flussonic web-interface supports setting up the transcoding for both streams and templates.

To set up the transcoder via the Flussonic web interface:

In **Media** > **Streams** or **Media** > **Templates** > click the stream or template you want to transcode. Then go to the **Transcoder** tab and click **Enable transcoder**.

Use the arrows on the right side of the page to expand or collapse groups of settings.



Audio encoding options

- **Copy from input** — select this to get the same audio characteristics as in the input stream.
- **Bitrate** — sets the audio bitrate.
- **Codec** (aac|opus|mp2a|pcma|ac3) — audio codec (the AAC codec is used by default).
- **Sample rate** (bypass|0|8000|16000|32000|44100|48000)
- **Channels** — sets the number of audio channels in an output stream.
- **Volume** — Output audio volume. It can be a value in dB (with "+" or "-") to be added to the input volume or a coefficient for multiplying the input volume. Learn more at the chapter How to change the volume level.
- **Split channels** — select this option if you want to split each audio track with multiple channels into several mono tracks.

Global options

These options apply to all output video tracks.

Device — specifies the transcoding device. For Flussonic Media Server, enables hardware acceleration and specifies the model and ID of the NVENC graphics card. Hardware transcoding allows for more streams to be transcoded on a single server. For Flussonic Coder, specifies the GPU used for transcoding the stream.

To automatically distribute many streams between GPUs, edit the configuration file and add the option `deviceid=auto` to `transcoder` for each stream.

Deinterlace — activates deinterlacing mode. Read the detailed description of deinterlace mode here. This field lists the methods available for the selected transcoder type (CPU, Nvidia NVENC, or others).

For Nvidia, this option represents two options in the configuration file (`deinterlace` and `deinterlace_rate`) that are used together. There are the following relations between the selected value in the **Deinterlace** box and the values of the options in the configuration file:

Deinterlace in UI	Options in file	Nvidia's deinterlace method
off	`deinterlace=false, deinterlace_rate=frame`	weave
on	`deinterlace=true, deinterlace_rate=frame`	adaptive
on double rate	`deinterlace=true, deinterlace_rate=field`	adaptive
adaptive	`deinterlace=adaptive, deinterlace_rate=frame`	adaptive
adaptive double rate	`deinterlace=adaptive, deinterlace_rate=field`	adaptive

Crop after decoding — with the majority of transcoder types, you can crop the video. Cropping allows you to get only part of the image area in the output stream. There are 4 figures with the following meaning: **Crop-X** and **Crop-Y** — the coordinates of the upper left corner of the output video image, as compared to the input image (that is, (0,0) is the upper left corner of the input video image), **Crop-Width** — the width of the output image, and **Crop-Height** — the height of the output image.

GOP size — sets the number of frames in a group of pictures (GOP). Learn more here.

Improve the transcoder performance by running it as part of Flussonic (use with caution). By default, the transcoder runs in a separate process from Flussonic: this is a more reliable choice. If you select the option **Improve the transcoder performance by running it as part of Flussonic** (or specify `external=false` in the configuration file), the transcoder will run in the same process as Flussonic Media Server. This mode speeds up encoding, especially when encoding audio or when using an Nvidia device. However, a transcoder error may cause Flussonic to crash.

> **Warning**
>
> When transcoding a number of streams on Nvidia NVENC, make sure that the option **Improve the transcoder performance by running it as part of Flussonic** has the same value for all the streams.

Video encoding options

There are three ways to add a video track settings to the transcoder:

- Click the button **Add video track** and (optionally) select a bitrate and height.
- To get the same output video characteristics as in the input stream, turn on **Copy from input**.
- Click **Duplicate** to add another track with the characteristics of the track that you have specified and want to duplicate.

Besides, you can copy entire transcoder settings to other streams.

After you have added a video track, you'll be able to edit its settings. To expand a track's transcoding settings, click the arrow.

All options are on a single screen:

- **Width** — the picture width in pixels on the display where it will be played by a player.
- **Height** — the picture height in pixels on the display where it will be played by a player.
- **SAR (X:Y)** — the proportion of the video display width. Learn more here.
- **Resize** — the strategy of resizing the video to the specified Height (and Width).
- **Background** — the color of the area in the player that is not occupied by the video after resizing. It is used only with the 'fit' strategy.
- **Bitrate** — specifies the bitrate of the video track.
- **Codec** (H.264|H.265|MP2V|AV1) — sets the video codec. The default value is H.264.
- **Profile** (baseline|main|high) — A specific codec-dependent profile of the output video. The profile allows to assume if the track can be played on a particular device.
- **Interlace** — used to get an interlaced stream from a progressive one. Learn more about the `interlace` option in here
- **Preset** — affects video quality and download speed. Read more about this option here.
- **B-frames** — the values 0|1|2|3|4 correspond to these sequences of frames: IP|IBP|IBBP|IBBBP|IBBBBP.
- **Open GOP** — allows an open GOP, meaning that the transcoder will divide an output stream into GOPs with slightly different number of frames, but close to the number specified in **GOP size**. This option applies only to encoding on CPU and it might help to reduce traffic a little bit.
- **Refs** — (reference frames) used in inter-frame compression to refer to frames that follow. For better quality, use more reference frames.
- **Level** — used for compatibility with old devices.
- **Logo**

To burn a logo into your video stream, specify the path to the file containing the logo and then choose where the logo will appear on the video. To use a single logo file for all output tracks, specify it in **Logo**, and the transcoder will resize it according to the size of each output video track.

Learn more about logo here.

- **Extended** — if the option you would like to add is missing on the screen, add it manually in **Extended**:

We recommend you to refer to the API reference for the full list of parameters with the relevant descriptions.

Saving or discarding your settings

To save the new values, click **Save**.

To delete all specified settings and turn off the transcoder for this stream, click **Disable transcoder**.

Copying the settings to other streams

To copy the settings to other streams:

1. Go to the **Transcoder** tab of a stream where you have already configured the transcoder settings
2. Click the button **Copy settings**
3. Go to the **Transcoder** tab of the stream where you want to apply the same settings and click **Enable and paste settings**. If the stream already had transcoder configured, the button will be **Paste settings**.

Configuring transcoder via the configuration file

Transcoding options can be specified in stream settings in the Flussonic configuration file `/etc/flussonic/flussonic.conf`.

To enable and configure transcoder via the configuration file, use the `transcoder` directive with a number of transcoder options.

When setting transcoder options, specify them in the following order:

1. Global options
2. Video track options (required and optional) for each track
3. Audio track options (required and optional)

See the example of configuring transcoder for the incoming stream `example`:

```
stream example {
  input fake://fake;
  transcoder vb=2048k size=1280x720 preset=slow ab=128k;
}
```

Here is an example of setting transcoding options to create a multibitrate stream:

```
vb=2048k preset=veryfast vb=700k size=720x576 preset=veryfast vb=300k size=320x240 preset=veryfast ab=128k
```

Transcoder options for anamorphic video

The Flussonic transcoder supports anamorphic video streams by taking the pixel sizes ratio into account. This was possible by giving the `size` parameter a new interpretation and by adding the new parameter `sar`.

Learn the detailed description of the options `size` and `sar` in the list of transcoder options on this page.

Apart from `size`, the parameters `aspect`, `force_original_aspect_ratio`, and `crop` were changed:

- `aspect` has been replaced with `sar`. Almost all transcoder types in Flussonic will interpret it as SAR (not DAR), the only exception is Nvidia NVENC.
- `force_original_aspect_ratio` is no longer necessary, and, if it is required, it is added automatically.
- The NVENC-only `crop` was added to almost all transcoder types in Flussonic (please don't confuse it with the resizing strategy 'crop').

The transcoder settings that you configured in earlier versions will stay the same and processed as previously. The transcoder processes the parameters in a new way only if you specify new parameters — SAR or the resize strategy (or both) — while no deprecated parameters (`force_original_aspect_ratio`) were specified.

Hardware transcoding

You can significantly increase the number of transcoded streams that the server can support by using a hardware transcoder.

Flussonic Media Server supports the Nvidia NVENC and Intel Quick Sync transcoding technologies.

One video stream can be transcoded using only one type of transcoder.

Read more about hardware transcoding in Hardware Transcoding with Nvidia NVENC and Intel Quick Sync Video.

Transcoder configuration options

The API schema for all transcoding options can be found in API Reference.

GLOBAL OPTIONS:

hw

`hw` — enables hardware transcoding. This option should be specified separately for each video stream.

deinterlace

`deinterlace` — activates deinterlacing, i.e., converting an interlaced video to a progressive video.

The interlaced video demonstrates even and odd scan lines as two individual fields. At first, the even lines pass on the screen and then the odd lines pass. Two of such even and odd scan line fields make one video frame. Interlaced videos are great for broadcasting as video images can be processed onto the screen with very little bandwidth. The drawback of interlaced video is that in fast motion, it may be blurred as only half of the image is captured at a time, movement along the frame causes motion artifacts.

Progressive video content shows the even and odd scan lines, that is the entire video frame on the screen at the same time.

Deinterlacing is necessary for comfortable viewing of legacy TV video on PC/mobile devices. It is specified once and acts immediately on all video streams.

The UI box **Deinterlace** corresponds to this option.

deinterlace_rate

`deinterlace_rate` — when encoding with Nvidia NVENC, you can remove duplicate frames that were produced after deinterlacing, preventing increased bitrate.

- `deinterlace_rate=frame` — from field sequence `1a 1b 2a 2b 3a 3b` we get frame sequence `1a1b 2a2b 3a3b`. The FPS stays the same,
- `deinterlace_rate=field` — fields `1a 1b 2a 2b 3a 3b` transform into `1a1b 1b2a 2a2b 2b3a` frames. The FPS increases two times after transcoding.

In case of using the Nvidia both options (`deinterlace` and `deinterlace_rate`) are added in the configuration file when you select some value in the **Deinterlace** box in the UI. There are the following relations between the selected value in the **Deinterlace** box and the values in the configuration file:

Deinterlace in UI	Options in file	Nvidia's deinterlace method
off	`deinterlace=false, deinterlace_rate=frame`	weave
on	`deinterlace=true, deinterlace_rate=frame`	adaptive
on double rate	`deinterlace=true, deinterlace_rate=field`	adaptive
adaptive	`deinterlace=adaptive, deinterlace_rate=frame`	adaptive
adaptive double rate	`deinterlace=adaptive, deinterlace_rate=field`	adaptive

crop

`crop` — crops the size of video.

Usage: `crop = x: y: width: height` :

- `x:y` — the coordinates of the upper-left corner of the output video within the input video.
- `width` — the width of the output video.
- `height` — the height of the output video.

gop

`gop=150` — sets the number of frames in a GOP. The encoder will create all GOPs of an exactly identical size — as specified in this option.

If you use encoding on CPU, you can use the `disable_cgop` option in addition to this option. It allows the transcoder to vary the GOP size slightly.

fps

`fps` — frame rate. Specified separately for each video stream.

VIDEO OPTIONS:

vb

`vb` (video bitrate) — specifies the video bitrate of the track. It is specified as a numerical value (1000k, 1500k, 2000k, etc). **The value must always end with *k***. Each `vb` option creates a new video track in the output stream.

The option `vb=copy` saves the parameters of the original stream, that is, it is simply copied to the outgoing stream.

> **Warning**
>
> When creating a multi-bitrate stream, we recommend that you don't use `vb=copy` and the numerical value `vb=NUMk` together. Otherwise, your viewers can experience issues while playing the video stream, such as glitching or stuttering.

preset

`preset` — the encoder *preset*, i.e., a set of values that determine a certain encoding speed, which influences a compression ratio. A slower preset will provide better compression (compression is quality per file size).

This means that, for example, if you target a certain file size or constant bitrate, you will achieve better quality with a slower preset. Similarly, for constant quality encoding, you will simply save bitrate by choosing a slower preset.

The list of supported presets:

- `veryfast`
- `medium`
- `slow`

The default preset is `medium`.

size

`size` — the size of output video on the display where it will be shown. The **size** option now includes width and height (in pixels), the resize strategy (crop, fit, scale), and background color. Thus, you can set the size in one of the ways: `size=WxH:fit:#aaFFEE`, size= `WxH:crop`, or `size=WxH:scale`.

The parameter `size` now means the size of a playback window on the screen rather than size in pixels. Previously, size was interpreted as pixel size, and the size of the playback window depended on a stream's SAR or on the value of the `aspect` parameter.

logo

`logo` — allows you to overlay a logo. The transcoder adds the logo before the video is resized as specified in the `size` option. This means that the logo can be visibly stretched if the size was changed significantly.

Learn more about adding a logo

alogo

`alogo` — allows you to overlay a logo. The transcoder adds the logo after the video was resized as specified in the `size` option. This measure prevents the logo picture from stretching that might occur when the `logo` option is used. You will need to prepare and specify a separate file with a logo for each size of the resulting video track.

vcodec

`vcodec` — allows you to set the video codec. The default value is H.264. Flussonic Media Server allows you to encode in H.265 (`HEVC`), `MP2V` (MPEG -2 Video File), or AV1. Specify this setting separately for each video stream.

It is not possible to use the `MP2V` when hardware transcoding.

`AV1` is supported for NVENC only.

refs

`refs` — the number of reference frames. This option should be specified separately for each video stream.

bframes

`bframes` — specifies the number of B-frames. When set to 0, this option disables b-frames. This may be necessary, for example, when broadcasting to RTSP. Specified separately for each video stream.

sar

`sar` — modifies video aspect ratio. Used for creating non-anamorphic video from anamorphic video. Has replaced the deprecated `aspect` but does not copy the old behavior.

SAR in Flussonic's terminology is the ratio of the width of the display representation to the width of the pixel representation of video. The width of the display representation is the number of pixels on the matrix of the display, this is what Flussonic passes to the player for playback. And the width of the pixel (internal) representation is the number of pixels in the original YUV.

In the UI `sar` appears in advanced video options.

For the transcoder on CPU and Flussonic Coder `aspect` is now processed as SAR — meaning the proportions of video display. For the transcoder on Nvidia NVENC `aspect` is interpreted as DAR (the ratio of player window horizontal and vertical sizes) and processed as in earlier versions of Flussonic.

> **Warning**
>
> Aspect ratio modification is not supported when transcoding with Intel QuickSync (the `hw=qsv` option).

Flussonic calculates the output video resolution based on `sar`. A video with the internal pixel width of 720 and sar 16:11 will have the display width of 1048. A picture of this width, in display pixels, will appear when the stream is played back in players.

force_original_aspect_ratio (deprecated)

`force_original_aspect_ratio=true` — keeps the original aspect ratio by adding black bars (letterboxing and pillarboxing.) This option is useful when you want to keep output resolution while switching between sources with different parameters.

disable_cgop

`disable_cgop=1` — allows an open GOP, meaning that the transcoder will divide an output stream into GOPs with slightly different number of frames, but close to the number specified in `gop`. This option applies only to encoding on CPU (it is not available on hardware transcoders) and it might help reduce traffic a little bit.

interlace

`interlace` — is used for making an interlaced stream from a progressive one.

The option takes values: `interlace=tff|bff|tff_separated|bff_separated|mbaff|true`

When the option is not specified in the config, the output stream is progressive. If you enable this option without specifying the encoding method (that is, if you specify as follows — `interlace=true`), then for producing interlaced video Flussonic will use the default method (the default method depends on the transcoder type). You can also specify another supported method.

- `tff` — interlaced, top field first, interleaved field store. This method is used with hw=qsv, nvenc.
- `bff` — interlaced, bottom field first, interleaved field store. This method is used with hw=qsv, nvenc.
- `tff_separated` — interlaced, top field first, separated fields. This method is used with hw=qsv.
- `bff_separated` — interlaced, top field first, separated fields. This method is used with hw=qsv.
- `mbaff` — interlaced libx264 MBAFF method. This method is used only with hw=cpu.
- `true` — enables encoding into interlaced video by using the default method for the encoder specified (`mbaff` is the default method for `hw=cpu`, `tff` is the default method for `hw=qsv`, `hw=nvenc`).

rc_method

`rc_method` is used for creating output video with constant bitrate suitable for broadcasting to television networks.

The option takes values:

- `rc_method=cbr` — the encoder will produce a DVB-C compliant stream.
- `rc_method=vbr` — do not encode a stream to be DVB-C compliant.

As for now, using this option consumes resources (one CPU core for one MPTS stream with CBR).

AUDIO OPTIONS:

ab

`ab` — sets the audio bitrate. This option should be specified only once, even if there are several audio tracks. The value must always end with *k*.

acodec

`acodec` — sets an audio codec. Accepts the following values: `aac`, `mp2a`, `opus`, `pcma`, `ac3`. By default, all audio tracks are encoded with AAC.

ar

`ar` — sample rate.

ac

`ac` — the number of audio channels.

avol

`avol` — output audio volume. It can be a value in dB (with "+" or "-") to be added to the input volume or a coefficient for multiplying the input volume. Learn more at the chapter How to change the volume level.

split_channels

`split_channels` — if this option is set to `true`, each audio track with multiple channels will be split into several mono tracks.

Burning text, time, and subtitles

SETTINGS

You can specify the burning settings in one or both of the following ways:

- As a global setting. This corresponds to transcoder.global.burn parameter in the API schema (placed before all `vb` in the config file). The text, time, or subtitles burned this way will be added to all tracks resulting from the transcoder.
- As a setting of a particular video track. This correspond to transcoder.video.burn API parameter (placed after corresponding `vb` parameter in the config file). The text, time, or subtitles burned this way will be added for that one track only. Track setting overrides the global setting.

General syntax for the burn option:

`burn=<filter>@<text:pos:x:y>@font:<ttf:size:color:alpha>@box:<border:color:alpha>`

Here:

- filter — time|sub|text
- text — subtitles track (for example t1) for `sub`, text (for `text`), %T or %F or their combination (for `time`).
- text:pos:x:y — pos — position letter (see below), x:y — offset to the right or left (x) and up or down (y) to the center of the screen. The offset cannot be negative.
- font: — ttf — font file TTF, color — font color, alpha — opacity (use values from 0.1 to 1.0, 0.0 — completely transparent, 1.0 — completely opaque).
- box: — border — the padding from the border of the box to the text in it, color — box color, alpha — opacity (use values from 0.1 to 1.0, 0.0 — completely transparent, 1.0 — completely opaque).

Short version of the burn option:

- `burn=time` displays the time in the default format
- `burn=sub` displays WebVTT subtitles from track t1
- `burn=text` displays an empty string (later you can set the text using the API)

Rules:

- The first group of parameters (`<filter>`) is required, other groups are optional (text, font, box).
- The first parameter in each group is required (text, font, box).
- The order of the parameters must be observed.
- Missing parameters will be replaced with default values: size — 16, color — `black` for `box` , `white` for `font` , border — 6, alpha — 0.8, ttf — FiraCode-Regular.ttf
- You can use several `burn` options of a different type (for example, `burn=text` and `burn=time`), but not the same type twice for the same track.

Below is a detailed description with examples.

burn=time

`burn=time` — burns the clock time. You can optionally specify time offset relative to the time of the Flussonic server, offset=0 by default.

You can customize the display of time by setting the font (font) and position on the screen (box) — see Text display settings.

Configuration examples:

- `burn=time@offset+3` — shows the time in the default format `YYYY-MM-DD HH:MM:SS` and in the timezone +3 hours relative to the Flussonic server time.
- `burn=time@%Toffset-3:tr@box` — shows the time in the format HH:MM:SS in the timezone -3 hours (relative to the Flussonic server time) in a dark box in the upper right corner (tr = top right).
- `burn="time@%F -- %Toffset-2:c@font:FiraCode-Regular.ttf:26:green:0.8@box:yellow:12:0.6"` — shows the time in the format `YYYY-MM-DD -- HH:MM:SS` and in the timezone -2 hours in the center of the screen, in FiraCode-Regular type and in a yellow box.

burn=sub

`burn=sub` — burns subtitles (dvb_teletext, dvb_subtitles or closed captions).

It is required first to extract subtitles and convert them to text in order to pass them to the transcoder for burning. For example, if the stream contains closed captions, use the option `cc.extract` .

You can customize the display of subtitles by setting the font (font) and position on the screen (box) — see Text display settings.

Option example:

```
burn="sub@t1:cb:10:10@font:Arial-Regular.ttf:30:white:1.0"
```

Here:

- `sub` — indicates that Flussonic will take subtitles (dvb_teletext, dvb_subtitles or closed captions) from the input stream and burn them into the output stream.
- `t1` — the number of a track containing WebVTT subtitles.
- `cb` (central bottom) — the location of subtitles.
- `10:10` — horizontal and vertical shift to the center relative to the specified location.
- `font:FONT_NAME.ttf` — the font. See the notes about font below the example.
- `30` — the font size.
- `white` — the font color.
- `1.0` defines the text transparency (use values from 0.1 to 1.0, 0.0 — completely transparent, 1.0 — completely opaque).

Example for a stream that contains closed captions:

```
stream example {
  input udp://239.0.0.1:1234 cc.extract;
  transcoder vb=3000k burn="sub@t1:cb:0:80@font:default:35:white:1.0" vcodec=h264 open_gop=false preset=veryfast size=1920x1080:scale:#000000 vb=1800k
burn="sub@t1:cb:0:80@font:default:25:white:1.0" vcodec=h264 open_gop=false preset=veryfast size=-1x720:scale:#000000 ab=128k;
}
```

In this example, we extract closed captions by using the option `cc.extract`. You may need to use other options instead of this one.

> **Note**
>
> Depending on what kind of subtitles your input stream contains, use the corresponding options to extract them to text format.

burn=text

`burn=text` — burns the specified line of text.

You can customize the display of the text line by setting the font (`font`) and position on the screen (`box`) — see Text display settings.

Option example for burning a text to a specific track:

```
transcoder vb=3500k burn=text@Hello:tr@box:green ab=64k;
```

FONT

- Flussonic supports .ttf font files.
- Flussonic looks for the specified font file in the subdirectory `font` of the `/etc/flussonic/` directory. This means you can place the font file like `/etc/flussonic/font/SomeFont.ttf`
- If the font file specified is missing in `/etc/flussonic/font/`, the default `FiraCode-Regular.ttf` font will be used, which is included in Flussonic.
- You can specify the full path to a font file. Make sure you put the font file in the directory you specified. For example, let's specify the path to one of the system fonts:

```
font:/usr/share/fonts/truetype/freefont/FONT_NAME.ttf:50:white:1.0"
```

- You can explicitly specify the `default` font: `font:default:30:white:1.0`. Flussonic will use `FiraCode-Regular.ttf`. However, if you copy any font file named `default` into the fonts directory `/etc/flussonic/font/`, Flussonic will use that file.

Examples:

`font:default:50` — the default font with size 50

`font:default:24@box` — the default font with size 24 in a box with default dimensions

`font:default:26:blue` — the default font with size 24 and color blue

`font:default:26:blue:0.9` — the default font with size 24 and color blue and 0.9 opacity (use values from 0.1 to 1.0, 0.0 — completely transparent, 1.0 — completely opaque).

LOCATION

Additionally, you can specify the location on the screen where the data will be displayed.

- tl — top left
- tr — top right
- bl — bottom left
- br — bottom rigth
- c — center
- ct — center top
- cb — center bottom

Together with these location abbreviations, you can specify the horizontal and vertical offsets from the specified location:

- `cb:10:200` — the text will be centered at the bottom of the frame with offsets x=10 (right) and y=200 (up)
- Offsets are 10 by default. Offsets can be positive integers or 0.

Example:

`burn=time@%F:cb:0:200@font:default@box` — time in the format YYYY-MM-DD centered at the bottom of the frame and with an offset of 200 upwards.

Note

For processing and displaying fonts, Flussonic uses the libfreetype library, which is included in the set of libraries provided in the flussonic-transcoder-base package. To render text in the CPU and Nvenc transcoders, the ffmpeg drawtext filter is used.

3.4.2 Transcoding

Satellite video is transmitted in either MPEG-2 or H. 264 (aka AVC or MPEG-4 part10). As a rule, MPEG-4 part 10 is for simplicity reduced to MPEG-4, but it is important not to confuse it with MPEG-4 part 2, which is absolutely incompatible and is not like H. 264; it was used in older IP cameras.

Audio is transmitted in MPEG audio layer 2 (abbreviated mp2) or in ac3 (a/52).

It is important to understand that today H264 is usually compressed with intra-refresh, i.e. a video stream contains no reference frames (IDR or keyframe). This compression method makes it possible to smooth out bitrate surges.

As a result, none of the transmitted satellite variants of audio or video can be played on iPhones. The browser would play back only H264.

During transmission via the Internet, video from MPEG2 can usually be safely compressed to H264 with a threefold decrease in traffic.

When transmitting HD channels via the Internet, one has to compress the stream into several qualities: from HD with the best quality to standard SD to compensate for overloaded channels.

In the end, in order to provide high-quality OTT service, the video from the satellite should be transcoded into other codecs and qualities.

It is important not to confuse *transcoding* with *repackaging*. Transcoding is a very resource-intensive operation that includes:

- unpacking the stream to encoded video/audio
- decoding to raw video/audio
- changing the size and other parameters
- reverse coding
- packing into the transport for the stream

Packing and unpacking are relatively easy operations; the streaming server can handle up to 1,000 channels on the same computer. The same computer can be used for transcoding 1 to 30 channels, depending on the size and capacity of the computer.

For transcoding, you can use specialized dedicated hardware: either a CPU or a video card (an external one or integrated into the processor).

We will not consider specialized devices, since most of them are either computers with special application software, or extremely expensive and highly specialized equipment, or even unreasonably expensive devices that are sold exclusively through the manufacturer's marketing efforts and do not allow achieving any significant results.

H.264

For video processing on the CPU there are several software applications, but only two libraries can be reasonably used for compressing into the H264 codec on CPU: a free libx264 and proprietary MainConcept. Everything else is either worse, or much worse, both in terms of the result and in terms of the use of resources.

Working with MainConcept will not be considered in this article, only libx264 will be mentioned.

Today, the H264 codec is de facto the standard for video, since it is supported by all modern devices, except perhaps for some devices from Google.

There are virtually no alternatives to it. Today there is a growing H265, it already has a lot of support, but until not working with it is investing into the future.

Codecs from Google: VP8 and VP9 are more Google's desire to pull the blanket over, rather than something actually useful. The resulting quality is worse, there is no support for hardware decoding, and therefore the price of the device grows.

When encoding video, one should understand that a balance should be observed between these parameters:

- delay inside the encoder in frames
- CPU usage (the number of milliseconds required for compressing a single frame)
- output image quality (pixel rate and color)
- output bitrate

For all kinds of broadcasts, CPU usage is absolutely critical. If the encoder settings require full CPU load or more, the video will fail to be encoded in real time, and therefore the streaming nature of the video will be lost.

VOD does not have such tight restrictions, and a one-hour long movie may be encoded for three hours if you wish to lower the bitrate. With that, for broadcasting video, usually the full CPU capacity is not used, in order to process 10 channels on the same computer, rather than 4.

As to the delay inside the encoder, it is critical for video conferencing, but is absolutely not critical for IPTV. Even a 5 seconds delay in TV broadcasting does not change the quality of service.

There is a clear relation between the bitrate and the quality of connection: the more information about the picture is transmitted, the better it will be displayed. The quality of the picture may be improved by reducing the bitrate, usually by selecting more efficient compression tools that require a greater delay and more cycles.

Understanding this complex relationship is needed for better understanding of the assertion that "our encoder is the best encoder in the world." The comparison should be made by at least 4 parameters, but in the end it all boils down to the price for one-time and monthly transcoding of a single channel with the desired quality and output bitrate.

Using Flussonic for transcoding

Flussonic has a separate transcoder package.

Flussonic can decode video from UDP/HTTP, MPEG-TS, RTMP sources, and encode it to multiple qualities and sizes.

This feature becomes useful when there is a need to play the video not only on set-top boxes, but on tablets as well: the choice of available codecs is significantly less considerable there, as compared to set-top boxes.

It should be noted that in order to play the video on an iPhone, even the H264 from a satellite should be transcoded, since for variable bitrate the satellite usually uses intra-refresh coding mode that creates videos that cannot be played back on iPhones.

Flussonic is more convenient than VLC or other variants of organizing transcoding, since it is controlled by a single configuration file and monitors the status of transcoding automatically. On the contrary, VLC requires writing many monitoring scripts for tracking the transcoding status.

The next important transcoding feature of Flussonic is automatic rebalancing streams if one of the servers goes down. If one of 20 transcoders fails at night time, the rest of the transcoders can be configured to automatically capture streams for transcoding, and the streamer will itself pick streams from the backup transcoders.

3.4.3 Flussonic Coder

Flussonic Coder is a hardware-software solution for video and audio transcoding that has the following advantages over other types of transcoders in Flussonic Media Server:

- allows large companies to cover customer needs comprehensively and predictably
- allows you to unify the technical support process
- helps integrators to protect projects
- keeps access to a subscriber device available

Flussonic Coder is a building block of the Flussonic Cluster required for processing, transmitting, and further video recording. It supports a video stream with plenty of formats, codecs, and protocols in any point of the Flussonic Cluster.

Flussonic Coder represents a server with our custom Linux OS, several NVIDIA Jetson modules and installed transcoding software. It's delivered with a firmware and doesn't require installing any other drivers. One NVIDIA Jetson module can transcode 6 Full HD streams to 3 profiles or 12 SD streams to 3 profiles.

The ingested video streams exist in the Flussonic Cluster as a sequence of elementary frames. Upon entering, the video is being de-multiplexed into atoms and on egress, the video is being multiplexed and packaged back for delivering in every modern video streaming protocol.

In this article:

- Configuring Flussonic Coder
- Configuring a stream to use Flussonic Coder

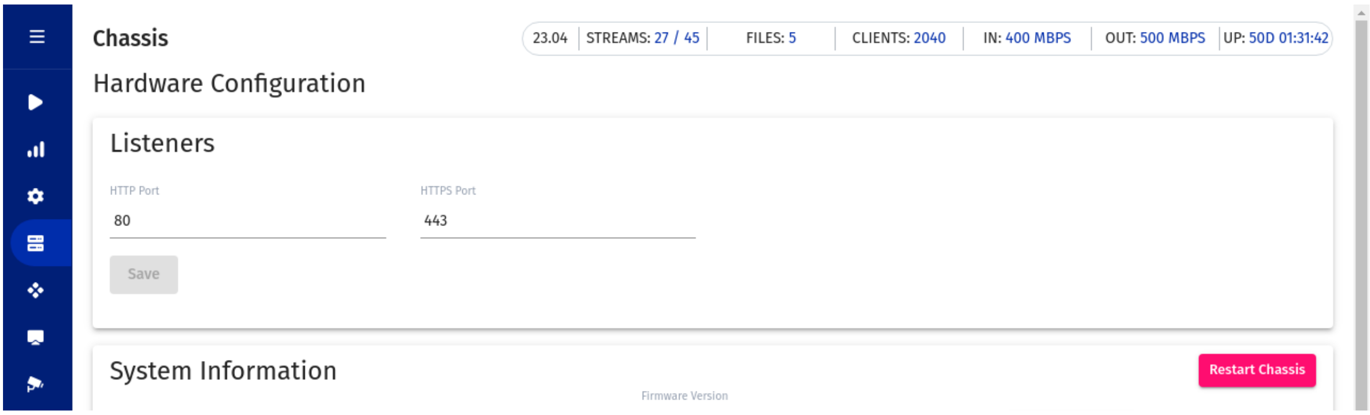
Configuring Flussonic Coder

Flussonic Coder has Flussonic Media Server installed, so you can configure its settings via Flussonic Media Server web interface. To view Flussonic Coder settings, go to the **Chassis** page in the side menu. Here you can see four sections described below.

SYSTEM INFORMATION

In the **System information** section, you can:

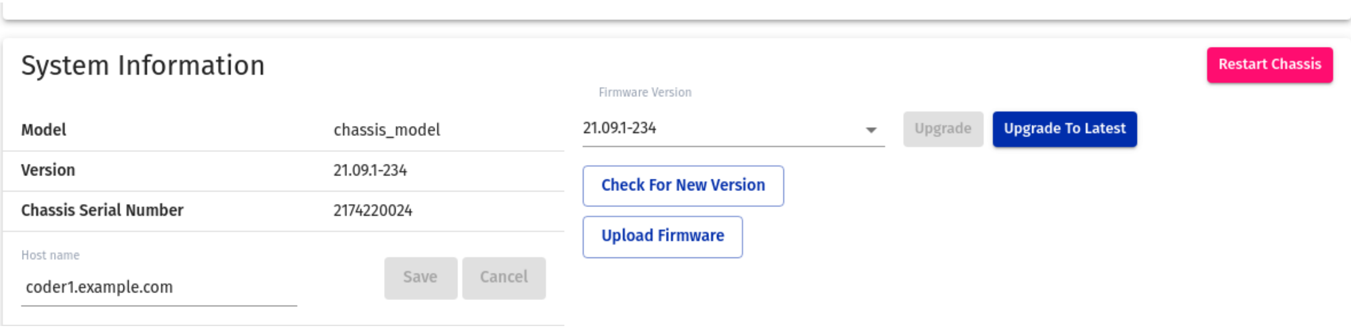
- view Flussonic Coder version
- view the firmware version and check for new version
- upgrade firmware to a specific version or to the latest version
- restart Flussonic Coder by clicking **Restart Chassis**



NETWORK CONFIGURATION

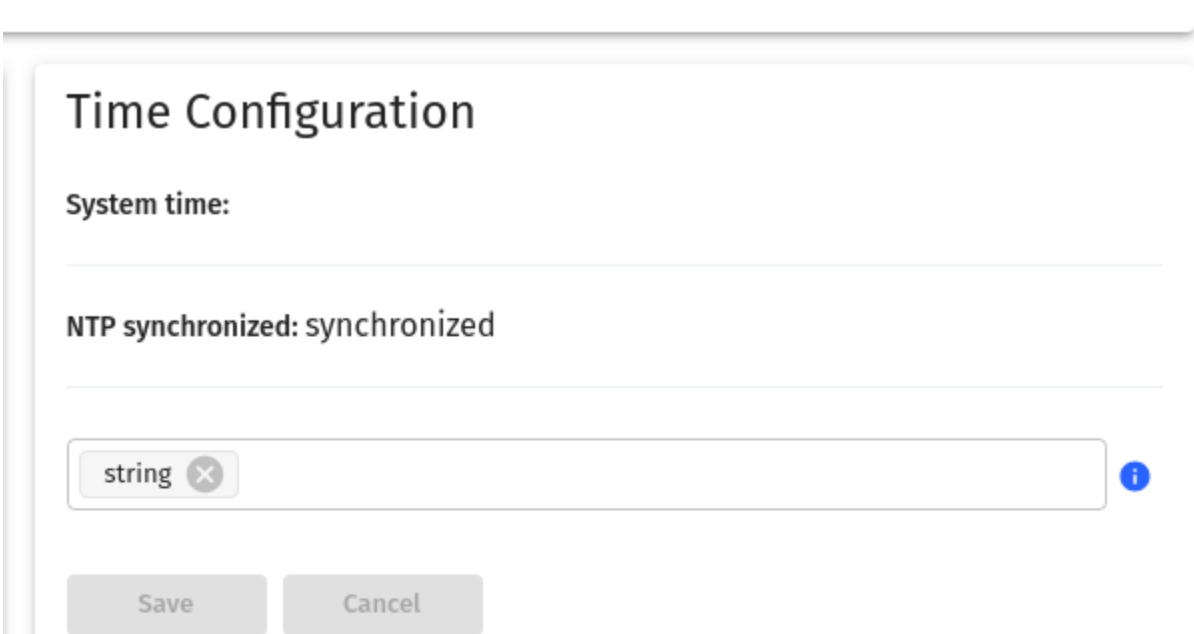
In this section you can change the default gateway interface. To do this, select the necessary gateway interface from the drop-down list in the *Default gateway interface* field and click **Save** to apply the changes.

> **Warning**
>
> The default gateway interface is used for checking license and updates and for sending responses to requests (API, HLS, etc.) received on the network interfaces of Flussonic Coder. Change the default gateway interface with caution and reach out to experienced network engineers to not lose control of your Flussonic Coder.



TIME CONFIGURATION

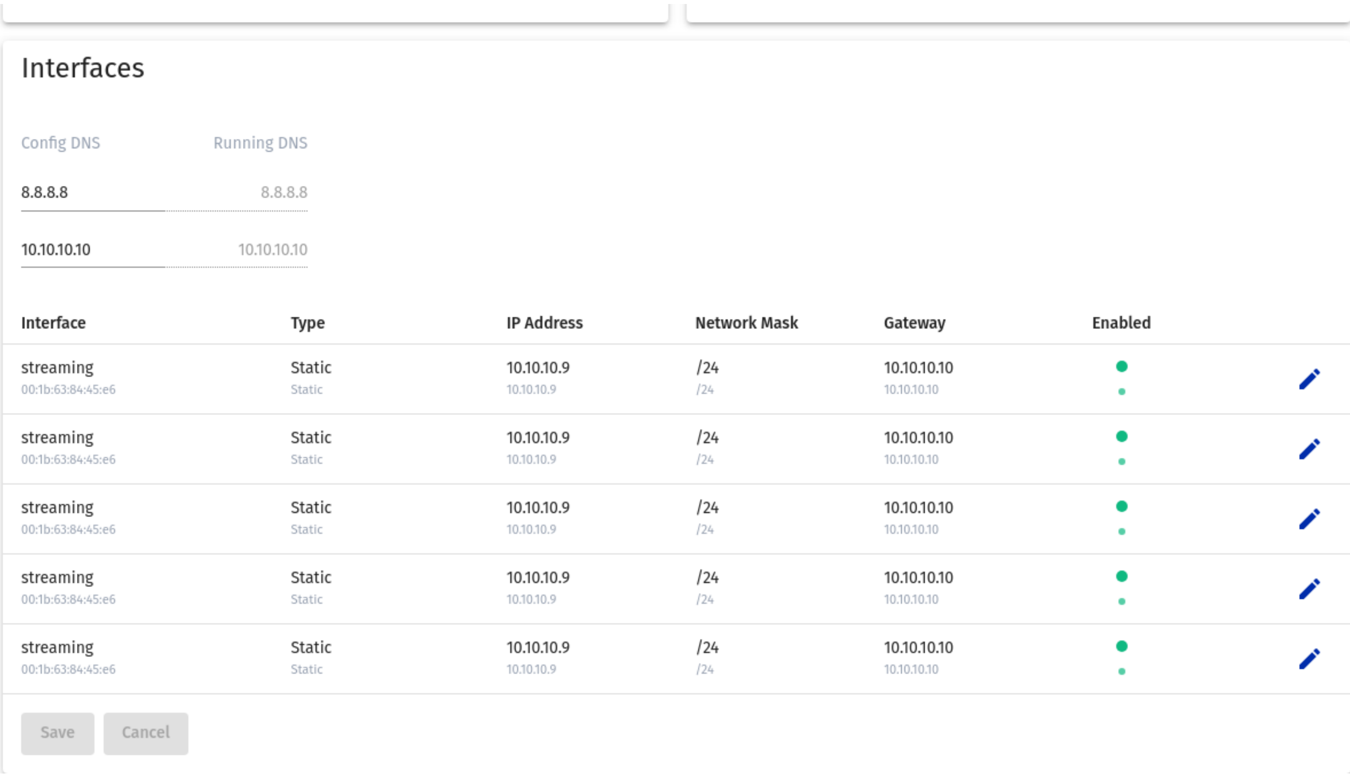
Here, you can check the current system time on the device and see the status of synchronization with NTP server. The URLs of NTP servers can also be added in this section.



INTERFACES

This section contains information about DNS and the list of all network interfaces used by Flussonic Coder. Here you can:

- specify IP address of DNS server
- select a type of an interface (static or DHCP)
- specify an interface IP address, network mask, or gateway

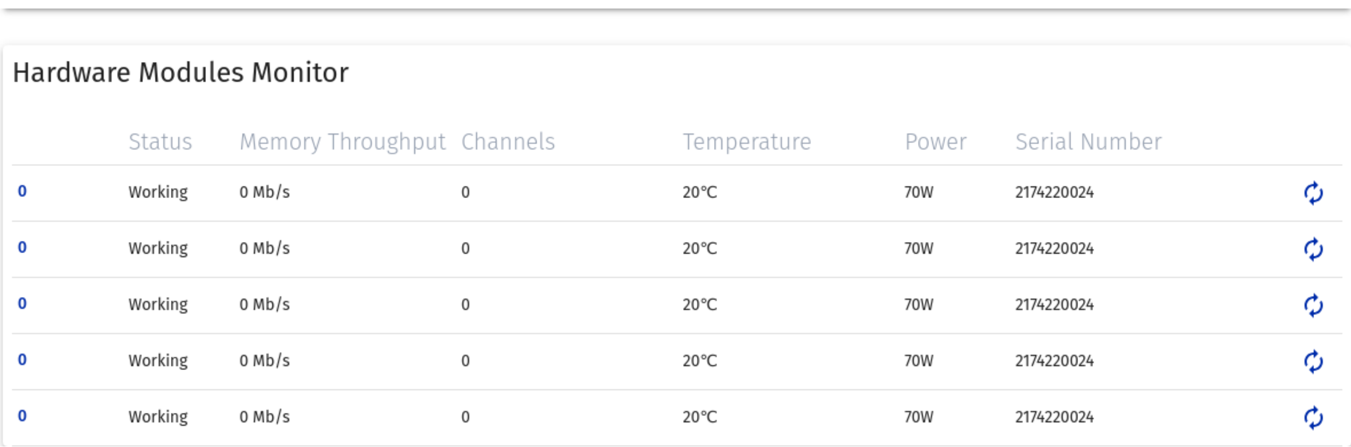


HARDWARE MODULES MONITOR

This section displays information about NVIDIA Jetson modules used in Flussonic Coder for transcoding. One NVIDIA Jetson module can transcode 6 Full HD streams to 3 profiles or 12 SD streams to 3 profiles.

Here you can:

- Monitor the following data on each module:
- running status
- number of channels a module is transcoding
- temperature
- power consumption
- serial number
- Reboot a module if necessary



> **Note**
>
> Please note that the position of NV02 modules in their slots in the chassis is strictly determined by the device architecture. The modules must be inserted into the chassis starting from the left slot, for example if you have one module then it must be inserted into the first slot on the left, two modules in first and second slot, etc. You cannot change the modules places, move or remove them.
>
> If you experience any problems with the numbering or some modules being unavailable, try checking that the modules are firmly fastened in their slots in the right order. Such issues may be due to loosing contact at transportation. Make sure to report the problem to our technical support team at support@flussonic.com.

Configuring a stream to use Flussonic Coder

To configure a stream to use Flussonic Coder for transcoding, use the `hw=coder` option in the `transcoder` directive in the stream's configuration. Learn more about transcoder settings on the Transcoder page.

Flussonic Coder supports the CUDA yadif method for deinterlacing video and allows to process dynamic scenes better. To use it, add `deinterlace=yadif` option to the configurationn file:

```
transcoder deviceid=0 hw=coder vb=10000k size=1920x1080 deinterlace=yadif ab=192k
```

3.4.4 Transcoding separate audio tracks

In some situations, it may be necessary to transcode input audio tracks separately, with different transcoding parameters. For example, when you capture video from a satellite, one audio track may be encoded with MP2A, and another one – with AC3. The AC3-encoded track has good quality and has not to be transcoded, however MP2A-encoded track should be transcoded to be used in browsers. Also, you may need to make two audio tracks with different parameters (e.g., bitrate) from one input track.

To transcode separate audio tracks, you can use the `atrack` option in the transcoder configuration. This option allows to specify an order number of an input track as an integer value or a string in `a<N>` format. For example, `atrack=1` or `atrack=a1` means the first input track.

All audio options specified in the configuration before the first `atrack` option are applied by default to all audio tracks. The options specified after the `atrack` option are applied to this particular track. If no options are specified after the `atrack` option, the output track will have the settings specified for all audio tracks.

Example for transcoding three input audio tracks with different parameters:

```
stream sample {
  input fake://fake;
  transcoder vb=1000k ab=copy acodec=aac atrack=1 ab=copy atrack=2 ab=64k atrack=3;
}
```

In this example, the first and the third input tracks will be transcoded with the original bitrate and the second track will be transcoded with the bitrate 64k.

Example for making two audio tracks from one input audio track:

```
stream fake {
  input fake://fake;
  transcoder vb=copy ab=64k acodec=ac3 atrack=1 ab=64k acodec=opus atrack=1;
}
```

In this example, transcoder will make two audio tracks from the first input audio track. The first output track will have the settings: `ab=64k`, `acodec=opus`. The second output track will have the settings: `ab=64k` `acodec=ac3` (as these settings are applied to all audio tracks).

3.4.5 How to Transcode a TV Channel into Multiple Qualities

In this article, we will review a typical transcoder configuration for an OTT service. In a separate article, we described in detail how such services work, but I will briefly remind you of the key features that require transcoding: - Transmission over the Internet: No guaranteed bandwidth to clients. - A wide variety of devices: TVs of different platforms and generations, smartphones, browsers. There is no universal protocol, DRM, codec, or resolution supported by all.

Transcoding into multi-bitrate will cover everyone: from 4K TVs connected to broadband internet to five-year-old smartphones watching a football match through 3G in a village.

Multi-bitrate is several tracks of different quality. Quality is a subjective parameter that consists of a combination of parameters: codec, resolution, bitrate.

How it looks for viewers:

- In normal mode, the viewer receives the highest quality that their internet and device allow. They do not notice the quality; it is already good for them.
- If the internet connection is unstable, the viewer notices that the picture is sometimes clear and sometimes "blurry." If the situation does not stabilize, they manually select a specific quality to prevent the player from "jumping" between different tracks.

With theory out of the way, let's get to practice.

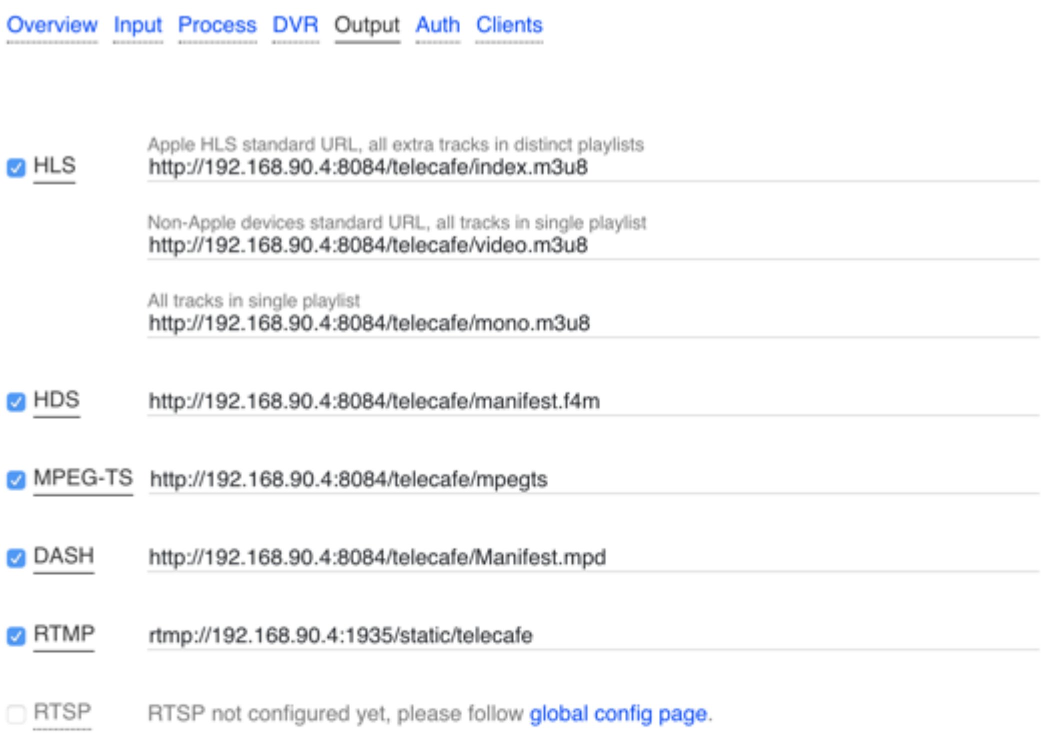
Preparation

You need a server with sufficient resources. We recommend our Flussonic Coder or a server with an Nvidia graphics card. How to choose a server for your service is described in a separate article.

Next, we assume that Flussonic Media Server is already installed, and the streams are captured. Most often, this will be UDP Multicast or still MPEG-TS, but from another source.

Configuration

- Open the **Transcoder** tab in the stream settings.
- Click **Enable transcoder** to show the configuration form.
- Open the dropdown menu next to the **Add Video Track** button and select several tracks. I will select all five: my source is 1080p, and I want the stream to be viewable even through a slow connection.
- Click **Save**.

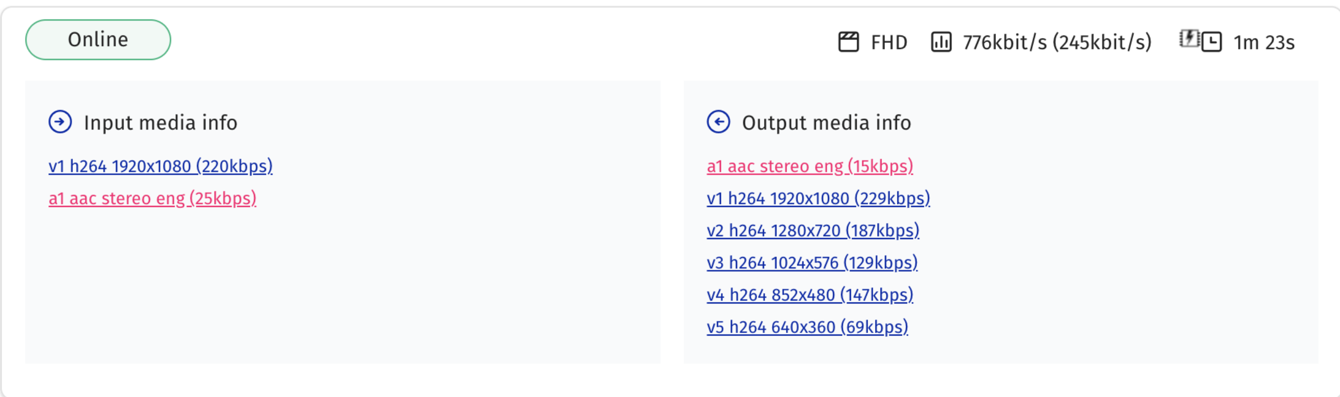


Advanced Configuration

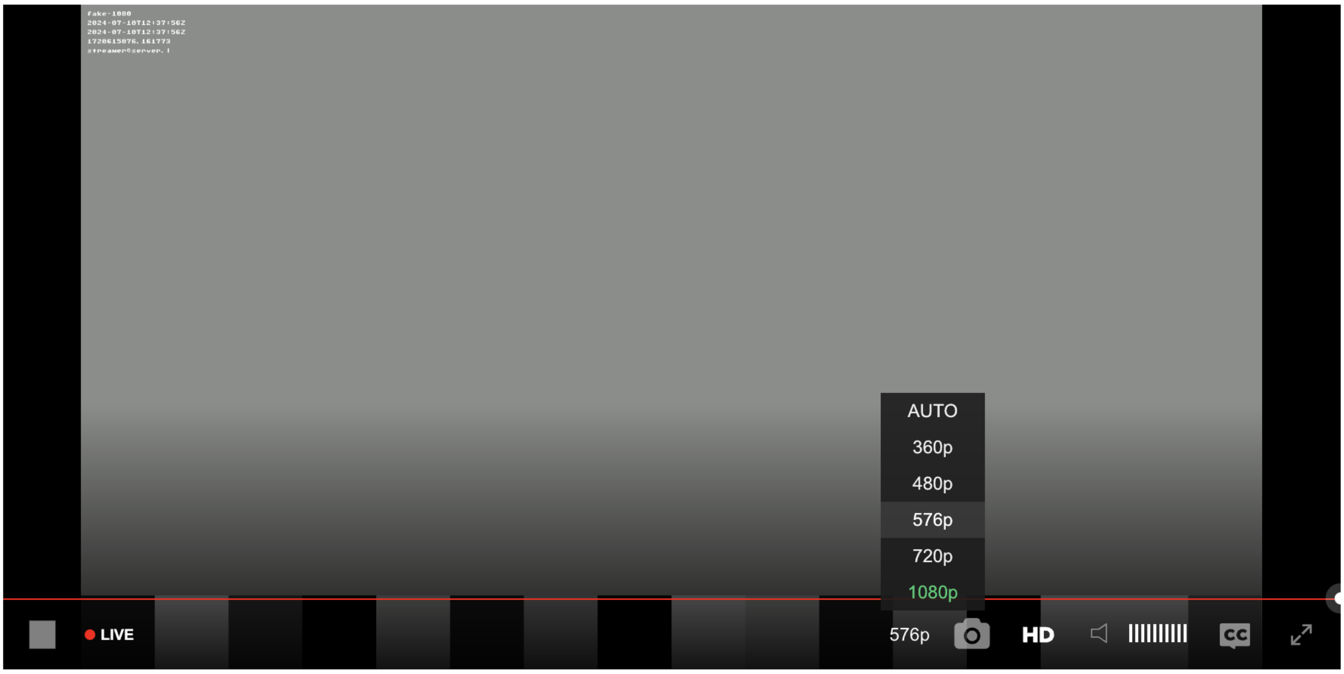
If our default settings do not suit you, you can override any parameter at your discretion. Detailed description of parameters.

How to Check that Multi-bitrate is Working

You can check the available output tracks in the **Overview** tab of the stream settings. If you have correctly configured the transcoder, you will see more tracks in the **Output media info** section than at the input.



And a quality selection menu will appear in the player.



Final words

Multi-bitrate is necessary for OTT service, and in Media Server, you can configure a typical configuration through the UI in a minute without delving into transcoding settings, simply by selecting pre-prepared settings. The configuration can be adapted to your quality requirements, regulatory requirements, or available resources.

3.4.6 Adding Streams from a Third-Party Transcoder

At the beginning of a project, you might already have a third-party transcoder that outputs multiple quality levels in UDP streams. How can these sources be integrated into Flussonic to ensure a smooth DASH/HLS output with reliable bitrate switching for the player?

The simple answer: you can't, replace the transcoder.

A more detailed answer will be provided in this article along with a debugging tool.

What are the problems with MBR UDP transcoders?

The most common problems that prevent creating a compliant MPEG-DASH stream from such transcoders are:

1. Different profiles have unequal GOP lengths. This issue also occurs when capturing from RTSP cameras of two profiles. This is fatal for most OTT IPTV players.
2. Keyframes are not synchronized. Different quality levels produced by independent transcoder processes, so one sets a keyframe at one frame, another at a different one. Synchronizing UDP sources by closely placed keyframes will result in desynchronization between videos (frames will disappear or duplicate during quality switching) and between video and audio. The audio will be taken from the first source, and the video from the second.
3. Timestamps are almost always unsynchronized. With UDP MPEG-TS, it is a normal practice to initialize the time with some arbitrary timestamp. As a result, after running a multibitrate UDP transcoder for a few days, you may end up with streams having timestamps that differ by many hours. When one stream restarts, everything goes out of sync. This could be partially mitigated by autocorrection, but see point 2.

Most multi-bitrate transcoders available on the market split video that must be synchronized into independent streams and then push those streams using a network protocol (UDP) that does not have guaranteed delivery or acknowledgment mechanisms. Even in a closed network environment, with minimal distance between the transcoder and packager, the use of UDP will eventually lead to the loss of transmitted data.

In this situation, packaging software must decide whether to drop one of the profiles from the manifest altogether, and stop offering this video quality for playout devices, or substitute the missing data with some placeholder (note that it should be in the same codec, resolution, and with the same GOP structure as the original so that the players won't freeze), or just produce TS chunks with zeros, knowing that it would result in freezes, visual artifacts, or even a closed playout session.

Dropping a single profile in a live broadcast might go unnoticed, especially if the dropped profile is not being played by the client. However, dropping a single profile will completely ruin the video archiving operation. When the media server writes the video into the archive for the purposes of nDVR or catch-up TV, it first initializes the archive for a certain type of MBR stream with a definite number of profiles, audio, and video tracks. If one of the profiles becomes temporarily missing, the integrity of the whole stream, including all other profiles, is also compromised, and there is no point in writing the remaining profiles to disk.

How to validate?

The package includes the utility `mbr_udp_analyze.erl`, which should be run in the command line with a list of addresses to be checked:

```
/opt/flussonic/contrib/mbr_udp_analyze.erl udp://239.150.12.1:1234 udp://239.150.22.2:1234 udp://239.150.12.3:1234
Synced 3 with shift 0.00
Synced 2 with shift 280.00
SYNC
Started stream analysis
Gop at 70228737.78 ERRORS:
#{error => not_starting_from_keyframe,input => 2,dts_shift => 40.0}

Gop at 70229737.78 ERRORS:
#{error => not_starting_from_keyframe,input => 2,dts_shift => 0.0}

Gop at 70230737.78 OK
Gop at 70231737.78 ERRORS:
#{error => not_starting_from_keyframe,input => 2,dts_shift => 0.0}
```

The log example shows that some segments can be assembled in a way that allows switching, but almost the entire stream is structurally unsuitable for OTT environments.

Here is what a good analysis would look like on an artificially assembled stream:

```
/opt/flussonic/contrib/mbr_udp_analyze.erl udp://239.150.12.1:1234 udp://239.150.13.1:1234 udp://239.150.14.1:1234
Synced 2 with shift 0.00
Synced 3 with shift 0.00
SYNC
Started stream analysis
Gop at 69999737.78 ERRORS:
#{error => not_starting_from_keyframe,input => 3,dts_shift => 40.0}

Gop at 70000737.78 OK
Gop at 70001737.78 OK
Gop at 70002737.78 OK
Gop at 70003737.78 OK
Gop at 70004737.78 OK
Gop at 70005737.78 OK
```

What to do?

It's unlikely that you have purchased a hardware transcoder that is capable of outputting a perfectly sincronized multibitrate video. Today, such devices practically do not exist, and hardware solutions at most imply the presence of hardware transcoding accelerator from Nvidia, Intel, or a few other solutions.

Most likely, you can easily repurpose your existing hardware by installing a regular Linux distribution and Flussonic on it, resulting in a guaranteed working MBR transcoding solution.

3.4.7 Hardware transcoding with NVIDIA NVENC

Transcoding video by using NVIDIA NVENC

Flussonic Media Server supports hardware video transcoding using the GPU on NVIDIA graphics cards. The list of supported cards can be found at NVIDIA website.

This feature requires the installed Nvidia driver of version 400 or higher.

The Flussonic transcoder can process 10-bit streams if you use Nvidia NVENC.

Transcoding of AV1 video is supported by NVIDIA Ada Lovelace or later generation cards only.

> **Note**
>
> Some Nvidia NVENC graphic cards have a limitation for a number of concurrent sessions (transcoded streams). If too many streams are transcoded with an Nvidia NVENC card, the corresponding alert message will be displayed in Flussonic UI. To check maximum of concurrent sessions for your type of a card, please refer to NVIDIA website. Also, if the card is 'unqualified', the limitation could be 'per server', not just 'per card'; please check the NVIDIA policy for that purpose.

INSTALLING THE DRIVER

Install the driver from the package:

Ubuntu 20.04:

```
apt-get install nvidia-driver-515-server --no-install-recommends
```

Make sure the `non-free` component is enabled in `sources.list`.

Official Nvidia driver downloads can be found on the Nvidia website. For help installing the drivers on Ubuntu, you may refer to the Ubuntu Knowledge Base.

To work with a lot of transcoder processes (more than 40), you'll need to increase the operating system's limit on open files. To do this, run the `ulimit` command:

```
ulimit -n 4096
```

And add the following lines to the `/etc/security/limits.conf` file:

```
* soft nofile 4096
* hard nofile 4096
```

ENABLING THE TRANSCODER

There are two ways to set up transcoding:

- In a stream's configuration entry, using the `transcoder` directive with various options.
- In the Web UI, under **Media** > choose a stream > **Transcoder**.

In both cases, you should add the `hw=nvenc` option to enable NVENC transcoding:

```
transcoder vb=2048k hw=nvenc ab=128k
```

SELECTING A CODEC

The default codec is H.264. When using NVIDIA NVENC, you can also use:

- H.265 (HEVC):

```
transcoder vb=2048k hw=nvenc vcodec=hevc ab=128k
```

- AV1:

```
transcoder vb=2048k hw=nvenc vcodec=av1 ab=128k
```

THE SUPPORT FOR 10-BIT COLOR STREAMS

If you use NVIDIA NVENC, the Flussonic transcoder can process 10-bit streams. This feature is supported for all input and output codec options.

Use the `pix_fmt` option ending at `p10` if you want to transcode a non-10-bit video to 10-bit. Please refer to the API reference for the full list of `pix_fmt` values.

For example, you can use this transcoder configuration to transcode 8-bit HEVC/H.264/AV1 to 10-bit HEVC:

```
transcoder vb=3000k vcodec=hevc pix_fmt=yuv420p10 ab=128k
```

Make sure that you have an up-to-date version of the NVIDIA drivers — 400 or higher. An Ubuntu version of at least 18.04 is also required.

SELECTING THE GRAPHICS CARD

Manual

If the system has multiple graphics cards, you can choose which one to use with the `deviceid=N` option:

```
transcoder vb=2048k hw=nvenc deviceid=1 ab=128k
```

The number of the card can be retrieved with the Linux console command `nvidia-smi`. By default, the first graphics card is used: `deviceid=0`.

Automatic

If you have a lot of streams, Flussonic will help you automatically allocate them between video cards for transcoding. Flussonic takes into account the GPU load and GPU memory consumption. With automatic allocation, you no longer have to determine that a GPU is overloaded and manually move streams to another card.

To enable automatic allocation of streams among GPUs, edit the configuration file and add the option `deviceid=auto` to `transcoder` of each stream:

```
transcoder vb=2048k hw=nvenc deviceid=auto ab=128k
```

CROPPING VIDEO

The option `crop=left:right:width:height` allows you to crop video:

```
transcoder vb=2048k hw=nvenc crop=0:0:100:100 ab=128
```

DECODING ON THE CPU

Decoding and encoding is performed on the GPU by default. To use the CPU for decoding, specify `hw=nvenc2` instead of `hw=nvenc`:

```
transcoder vb=2048k hw=nvenc2 ab=128k
```

DEINTERLACING

Deinterlacing is enabled by default when using `nvenc`. Additionally, you can specify a certain method with the `deinterlace` option. For example, add `deinterlace=yadif` to apply the **CUDA yadif** method when transcoding a stream:

```
stream test {
   input file://vod/test.ts;
   transcoder vb=4000k ab=128k deinterlace=yadif hw=nvenc;
}
```

All methods that you can use on NVIDIA Nvenc can be found in the UI in transcoder settings for a stream, in **Deinterlace mode**.

When using `nvenc2` (using the CPU to decode), deinterlacing has to be turned on explicitly with the `deinterlace=yes` option.

To disable resource-consuming deinterlacing, specify `deinterlace=0` in the transcoding settings.

Other parameters, such as `size`, `preset`, `bframes`, `level` are used in the same manner as the CPU transcoder options.

The `preset` parameter can have one of these values: `veryfast`, `medium`, `slow`.

READING TELETEXT FROM VBI IN SDI WITH NVIDIA NVENC TRANSCODING

Flussonic can read teletext from VBI in SDI input when transcoding SDI using NVIDIA NVENC. See Configuration examples for reading teletext from different sources for configuration examples.

Statistics on NVIDIA performance

You can collect statistics on the operation of Nvidia GPU if you enable saving statistics in the Pulse database. To start saving data, add the following directive to the Flussonic configuration file:

```
nvidia_monitor true;
```

To stop saving statistics on Nvidia, update the configuration file:

```
nvidia_monitor false;
```

You can always check you monitoring in the Retroview service provided to you in client area.

3.4.8 Intel Quick Sync Video

Information about supported platforms can be found on Intel's official GitHub page: https://github.com/intel/media-driver#supported-platforms

After installation, the QSV transcoder is available via the **hw=qsv** option:

```
stream example {
  input udp://239.1.1.10:5500;
  transcoder vb=3000k hw=qsv ab=64k;
}
```

Learn more about transcoder configuration.

Installing QSV on Ubuntu

We have prepared a .deb package that allows you to easily install the QSV drivers on **Ubuntu 18.04**.

```
apt install linux-base flussonic-qsv intel-media-va-driver libdrm-intel1 vainfo i965-va-driver libpciaccess0
reboot
```

Installing QSV on CentOS

Please follow Intel's official manual on the subject: https://github.com/Intel-Media-SDK/MediaSDK#system-requirements

3.4.9 Adding a logo when transcoding a stream

Flussonic Media Server can add a logo to your video in the process of transcoding:

When using the transcoder to overlay a logo on a video stream, the image will be "burned" into the video track. This means that it will be displayed on any device and recorded in the DVR archive.

Example:

```
vb=2048k preset=veryfast logo=/storage/logo.png@10:10 ab=128k
```

> **Warning**
>
> You should add a logo only with .png extension.

Here, `10:10` are the coordinates with an offset of 10 from the top left corner of the screen.

To place the logo in another part of the screen, you will need to use a slightly more complex formula.

To place the logo in the center:

```
vb=2048k logo=/storage/logo.png@(main_w-overlay_w-10)/2:(main_h-overlay_h-10)/2 ab=128k
```

To place the logo in the bottom left corner:

```
vb=2048k logo=/storage/logo.png@10:(main_h-overlay_h-10) ab=128k
```

To place the logo in the top right corner:

```
vb=2048k logo=/storage/logo.png@(main_w-overlay_w-10):10 ab=128k
```

To place the logo in the bottom right corner:

```
vb=2048k logo=/storage/logo.png@(main_w-overlay_w-10):(main_h-overlay_h-10) ab=128k
```

> **Warning**
>
> Adding a logo is possible only when you use the CPU or NVENC transcoder.

3.4.10 Overlaying dynamic text

To augment the frame with some unremovable dynamic text that will be displayed on any device both in live and archive, use Flussonic Media Server's transcoder. For example, the text may be:

- Name of a film or TV show.
- Weather forecast.
- Football match score.
- News feed or emergency notification.

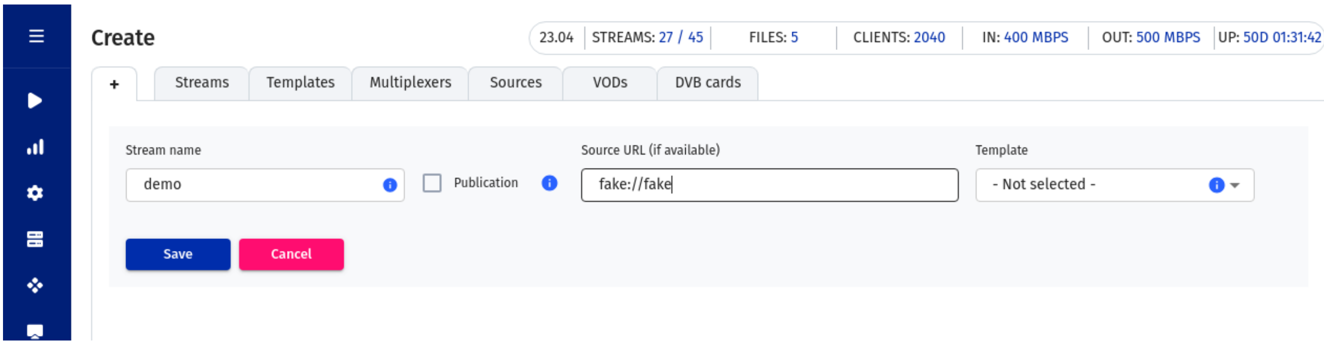
Note

An alternative to the transcoder is adding text on a player side. This method does not additionally load the server, but the text added in this way is: displayed differently or not displayed at all on TVs, mobile devices, set-top boxes, etc.; easily removed from the video; not available in the archive.

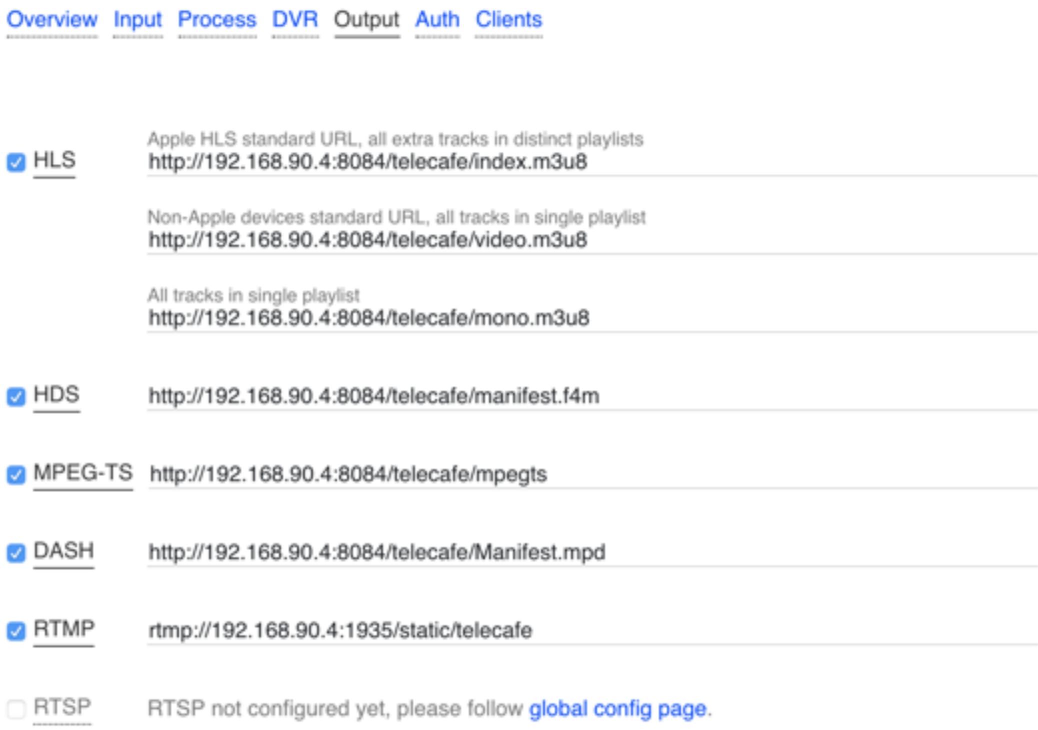
How to add the text

Let's proceed with a test example:

1. Add a test stream with `fake://fake` input to the Media Server configuration.



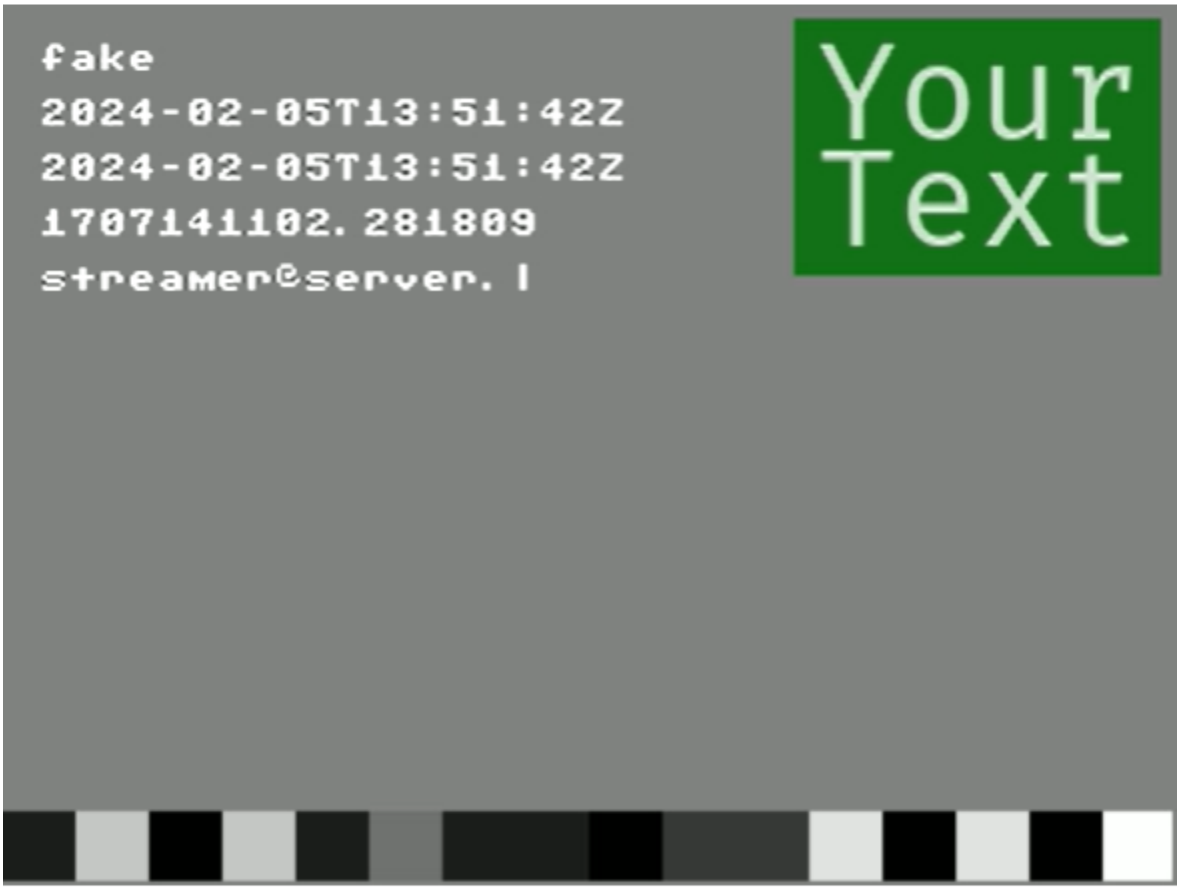
2. Enable transcoding on the **Transcoder** tab in the profile of the created stream.



3. Send the transcoder.global.burn option to the Media Server via API:

```
curl -u LOGIN:PASSWORD -X PUT "http://FLUSSONIC-IP/streamer/api/v3/streams/demo" \
-H "Content-Type: application/json" \
--data '{"transcoder": {"global":{"burn":{"text":{"position": "tr","text": "Your\nText","y": 10,"x": 10,"box": {"color": "green"}}}}}}'
```

Below is the result.



The procedure for changing text must be implemented in an external application or script.

Please refer to Burning text, time, and subtitles for details on the `burn` option configuration.

Related objectives

- Adding a logo when transcoding a stream about adding a static image over the video.

3.4.11 How to change the volume level

If one or a few of your sound sources have higher or lower volume level than the others you might want to adjust it. There are two ways to do that: through *Flussonic* configuration file or through *Flussonic UI*. We will provide you with both and you will choose the one that suits you best.

The value can be specified in decibels (dB) or it can be an integer/float (3, 0.5, etc.). By default it equals to 1.

- If it is just an integer or a float, the output audio volume is calculated by this formula:

*output_volume = avol * input_volume*

- If specified in decibels (dB), the output audio volume is calculated by slightly different formulae:

output_volume = input_volume +/- avol

depending whether it is a positive (+9dB) or a negative value (-6dB).

Note: Do not forget to use plus ("+") or minus ("-") when specifying the value!

Through Flussonic configuration file

To change the volume level of the transcoded stream in *Flussonic Media Server* you have to add the parameter `avol` in the description of the stream in the configuration file (`/etc/flussonic/flussonic.conf`) as follows:

```
stream example1 {
  input udp://239.0.0.1:1234;
  transcoder vb=copy ab=128k acodec=aac avol=2;
}
```

By default `avol=1` . In the example above we increase the volume level by 2: `avol=2` . Then if you specify `avol=0.5` , it will be halved:

```
stream example2 {
  input udp://239.0.0.1:1234;
  transcoder vb=copy ab=128k acodec=aac avol=0.5;
}
```

The following example shows the value specified in decibels (dB) that reduces the original value by an amount of 6 dB:

Note: Do not forget to use plus ("+") or minus ("-") when specifying the value!

```
stream example3 {
  input udp://239.0.0.1:1234;
  transcoder vb=copy ab=128k acodec=aac avol=-6dB;
}
```

Through Flussonic UI

You can change volume level via the *Flussonic UI*:

1. Open the *Flussonic UI*.
2. Go to **Media** -> **Streams** and click on the name of the stream you would like to change the volume level of.
3. Go to **Transcoder**. In the **Audio** settings, set the value in the *Volume* section. By default it equals to.
4. Save the changes by clicking **Save**.

Now you know how to change the volume of the transcoded stream in *Flussonic Media Server*.

3.5 DVR

3.5.1 Digital Video Recording (DVR)

Flussonic Media Server allows recording and playing video streams. This functionality is called DVR (digital video recording).

DVR records streams after transcoding and before DRM, meaning the processed but not yet encrypted video is saved to the disk. To record original, non-transcoded video, you need to use stream duplication (the `copy` protocol) and record the original stream, then transcode it separately.

The archive subsystem is a highly developed, reliable, and efficient technology with the following features:

- A unified data format for different playback protocols, allowing you to download once and distribute in various protocols
- Multi-level indexing, enabling efficient operation with year-long archives from the start without blocking at startup
- Parallel disk access, protecting the entire server from overloading a single device
- Efficient RAM cache management
- Integrated management of the data and their indexes, eliminating the need for additional database administration
- Precise frame timestamp preservation, necessary for integration with external analytics

More details about the archive features:

Archive Recording

- Expandable recording to local disks and cloud storage via S3 protocol. NFS is also supported yet highly unrecommended.
- Automatic deletion of old archives with precise granular retention of necessary episodes

Clustering

- Archive duplication to a neighboring server
- Transparent caching of the archive on a relay server

Playback

- Instantly available playback through various protocols and web interface: HLS, MPEG-TS, RTSP, RTMP, DASH
- Export of archive recordings to MP4 file
- Timelapse export
- Delayed viewing in another time zone
- Integration with IPTV middleware for viewing recorded broadcasts (Catchup TV)
- DVR API
- Video screenshots and saving them in the archive

3.5.2 DVR playback methods

You can view recordings by using the administration web interface or by embedding our DVR player on a web page.

An analog of the player that you see in the web interface can be embedded into your site by using the special embed.html address with the `dvr` parameter.

Also, you can access recordings via various video protocols by using special URLs.

Accessing DVR recordings by special URLs

URLS FOR DVR PLAYBACK

To access a recording by URLs, you can use stream mode or file mode.

A file, compared to a stream, has an end. That is, when playing a file, a player shows a timeline, and the video is limited (it has the beginning and the end). When playing a stream, a player doesn't show progress on the timeline, because the end of a stream is not known.

You can see this difference in URLs too. For example, a file's URL ends with "index-1345345345354-3600.m3u8" (the limits are defined: beginning at 1345345345354 and end after 3600 seconds), and a stream URL ends with "timeshift_abs-1345345345354.ts" (only the beginning is defined).

The URLs depend on the protocol that you use for accessing the DVR.

ELECTRONIC PROGRAMME GUIDE (EPG)

DVR can be used with EPG. The modern approach to the provision of television archive is to record the entire video, and then provide access to the archive (or rewind current video) using the EPG.

All metadata will be stored in a middleware and Flussonic Media Server will provide access to this archive as an infinite tape (with convenient navigation).

There are two modes:

- viewing already recorded video
- viewing live streams

If the broadcast is already over, the middleware forms the link based on the EPG to view the archive. The user can see the recorded movie as a normal file. For example, if the show is started at 18:15 Dubai time (14:15 UTC) on August 27 and continued for an hour, the middleware should create URL like this:

```
http://FLUSSONIC-IP:PORT/STREAM_NAME/index-1409148900-3600.m3u8
```

If the broadcast is not over, the middleware may create special URL to the archive, that allows to rewind live to its beginning.
Unfortunately this feature is supported by few devices and STBs, but nevertheless it exists. The URL for this unfinished broadcast will be like this:

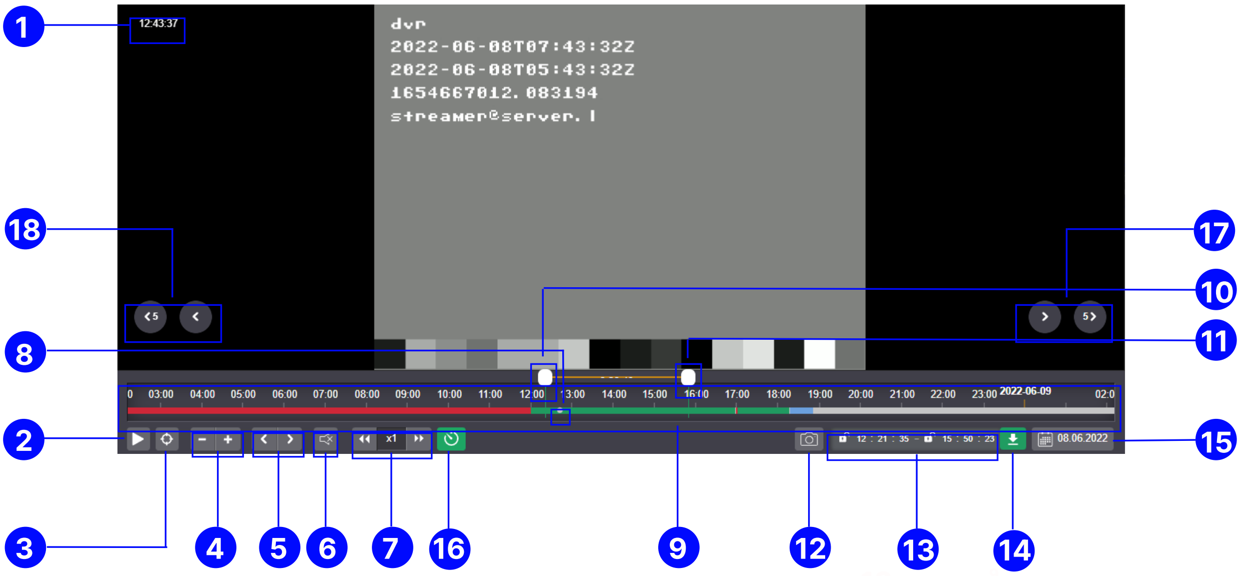
```
http://FLUSSONIC-IP:PORT/STREAM_NAME/index-1409148900-now.m3u8
```

Viewing the DVR recordings from the web interface

You can see the recordings of the DVR archive in the *Flussonic UI*. To do that:

1. Go to the stream settings by clicking on the stream name on the *Media* tab.
2. Then go to the **DVR** tab.

You'll see the DVR media player:



1. Current playback time of the recording

2. Start/pause button

3. Align the slider to the center of the timeline

4. Zoom in and zoom out the timeline

5. Go forward/backward along the timeline

6. Adjust the playback volume

7. Adjust the playback speed

8. Playback slider

9. Timeline bar

10. Start marker of a segment to export

11. End marker of a segment to export

12. Take a screenshot of the recording

13. Initial and final time of the fragment

14. Download the segment

15. Calendar

16. Seek per frame

17. Navigate to the next frame or 5 frames forward

18. Navigate to the previous frame or 5 frames back

You can also open the player in a new tab/window using the URL like the following:

```
http://FLUSSONIC-IP:PORT/STREAM_NAME/embed.html?dvr=true
```

This is the embed.html player for embedding video to a web page.

You can also play an archive in the Preview Player directly in the *Flussonic UI*.

NAVIGATION

The timeline bar (9) consists of several zones indicated with different colors: **red color** means no recording at this time, **green** means that a video record exists, **blue** means the current hour, and **gray** means the future.

You can click anywhere on the timeline bar or move the playback slider (8) to begin playing the video starting from that point. You can also use the following buttons to browse the timeline:

- aim sign (3) for aligning the playback slider to the center of the timeline.
- `-` and `+` (4) for zooming a time period, so you can select a time more precisely.
- `<`, `>` (5) for moving forward and backward along the timeline to an earlier or later time than displayed on the screen (you are not moving the video itself).
- calendar (15) to pick the date to watch or export the record for this day (if the record exists and the date does not go beyond the depth of the archive you defined).

You can also find a particular moment within an archive by seeking per frame. This can be useful, for example, when wathcing a recording from a surveillance camera and seeking for somebody's face or a car's license plate. To do that:

- Place the slider (8) in the green zone, where you expect the necessary moment to be found.
- The **Seek per frame** button (16) will become available. Click it.
- Use the buttons for navigation to the next frame or 5 frames forward (17) and to the previous frame or 5 frames back (18) to seek for the necessary moment.

EXPORTING DVR RECORDINGS

You can export one or more segments that cover the desired period recorded in the file and export the clips in MP4 format. To do that:

1. Move the start and end markers (10 and 11) to the start and end time points of the desired segment, or type in the time of the selected interval (13).
2. Lock them by clicking on the padlock icon (13).
3. Save the file by clicking the download button (14).

Learn more about DVR export.

3.5.3 Manage DVR with API

Flussonic provides a set of API calls to work with DVR. It allows you to automate the work with DVR in your system. In this article, we will overview the API calls to work with DVR and learn how to use them.

Overview

Flussonic allows you to obtain data about recorded streams and to set up stream recording in the archive. Some calls are available for administrators only, and some are available for end users.

For example, only the administrator can change the configuration or save a file locally on the server. End users can request information about a stream, JPEG thumbnails, etc.

Here is the list of available commands to work with DVR through the API.

ADMINISTRATOR-ONLY COMMANDS

- Configure DVR for stream
- Protecting DVR sections from deletion
- Recorded DVR ranges
- Export DVR segment to MP4 file

INFORMATION AND FILES AVAILABLE FOR END USERS TO REQUEST

- Requesting JPEG thumbnails
- Generating JPEG thumbnails on demand
- Requesting MP4 video thumbnails
- Download the DVR segment to MP4 or MPEG-TS file to a local computer

Administrator-only commands

Administrator-only commands must be authorized. The administrator should provide the basic token or bearer token when sending an API request. In the examples below we will be passing user credentials to access the *Flussonic* server — username and password — divided by a colon `username:password` to authorize the requests.

CONFIGURING DVR FOR A STREAM

To configure DVR for a stream, use the `Flussonic-API: PUT /streamer/api/v3/streams/{name}` call passing the desired JSON formatted DVR configuration:

```
curl -u username:password -X PUT "http://FLUSSONIC-IP/streamer/api/v3/streams/STREAM_NAME" \
> -H "Content-Type: application/json" \
> --data '{"dvr": {"root":"/storage","expiration":3600}}'
```

Here:

- `/storage` — directory where the DVR recordings will be stored
- `3600` — archive depth, or the retention period after which the DVR recordings are removed from the directory

To modify the existing DVR configuration for a stream, use the same call:

```
curl -u username:password -X PUT "http://FLUSSONIC-IP/streamer/api/v3/streams/STREAM_NAME" \
> -H "Content-Type: application/json" \
> --data '{"dvr": {"root":"/storage","expiration":172800}}'
```

Here:

- `/storage` — directory where the DVR recordings will be stored.
- `172800` — archive depth, or the retention period after which the DVR recordings are removed from the directory.

PROTECTING DVR SECTIONS FROM DELETION

Flussonic allows you to protect certain parts of the archive from automatic deletion. The *episodes* mechanism is used for that; an ***episode*** is a set of metadata about an archive section. All data about the episodes is stored in *Flussonic Central* and requested when it's time for archive cleanup. *Flussonic Media Server* doesn't delete archive sections with corresponding episodes, unless:

- Disk space reaches its limit in bytes or percentage.
- Episode expires.

> **Note**
>
> If you delete the stream with episodes or the episodes database becomes unavailable, *Flussonic* keeps the archive sections with episodes till the episode_expiration period ends to protect these files from being suddenly deleted.

To use the below API requests, install and configure *Flussonic Central*.

> **Note**
>
> You can use your own configuration backend instead of *Flussonic Central* to manage the episodes, see Managing episodes.

To create or update an episode, send the `Central-API: PUT streamer/api/v3/episodes/{episode_id}` request to *Flussonic Central*.

```
curl -u username:password --request PUT 'http://127.0.0.1:9019/streamer/api/v3/episodes/1' \
-H "Content-Type: application/json" \
--data '{"episode_id": 1,"media": "test-879c595839","opened_at": 1686808712,"updated_at": 1686808712,"closed_at": 1686823112}'
```

When it is no longer necessary to keep the protected archive recording, delete the episode with `Central-API: DELETE streamer/api/v3/episodes/{episode_id}`.

```
curl -u username:password --request DELETE 'http://127.0.0.1:9019/streamer/api/v3/episodes/1' \
```

> **Note**
>
> Note that username and password that you should use for authorizing API requests to *Flussonic Central* are not the same as *Flussonic*'s username and password. Check *Central*'s credentials in `/etc/central/central.conf`. The auth token is base64-encoded string `username:password`.

You can also view a list of episodes and information about a specific episode. The corresponding requests are provided in the API schema.

RECORDED DVR RANGES

Flussonic can provide a list of DVR recorded ranges for a stream (how many intervals with recordings there are, the start of the interval and its duration).

To request a list of DVR recorded ranges, use the `Flussonic-API: GET streamer/api/v3/streams/{name}/dvr/ranges`, where `{name}` is a stream name:

```
curl -u username:password http://FLUSSONIC-IP/streamer/api/v3/streams/STREAM_NAME/dvr/ranges
```

> **Note**
>
> All DVR ranges are returned in UTC timestamps.

If you need to manually remove a DVR recording in a specified range for a stream, use `Flussonic-API: DELETE streamer/api/v3/streams/{name}/dvr/ranges`, where `{name}` is a stream name:

```
curl -u username:password -X DELETE http://FLUSSONIC-IP/streamer/api/v3/streams/STREAM_NAME/dvr/ranges \
> -H 'Content-Type: application/json' \
> --data '{"from":1675326686, "duration":7200}'
```

Here:

- `1675326686` — start time in Unix time
- `7200` — duration in seconds

EXPORT DVR SEGMENT TO MP4 FILE

Admin is allowed to export DVR segments to MP4 files and upload them to a remote location, directly to the server hard drive or the Amazon S3 bucket.

MP4 export solution

The main problem when exporting an archive to a file of any format is that you need to specify file size in the file header. There are several solutions to this problem, for example, a software can create a temporary file on disk or make two passes through the archive (locate the fragments on the first pass and create a file on the second).

Flussonic Media Server exports the archive to fragmented MP4 in one pass without any temporary files thanks to the features of fMP4 format. The header of an fMP4 file does not contain information about frames, so it can be generated instantly. The data itself is divided into small independent fragments, each with its own header which contains information about the frames.

Thus, loading of the exported fragment begins immediately, even if the file has not yet been generated or even recorded. And if the export is interrupted, for example due to a disconnection, the already downloaded part of the file can still be viewed.

> **Note**
>
> There are other ways to access the archive besides downloading. Try DVR playback instead of export.

Export DVR segment to MP4 file via a URL

To do that, use the `Flussonic-API: POST streamer/api/v3/streams/{name}/dvr/export`, where `{name}` is the stream name.

Required parameters:

- `{from}` — start time of the DVR segment in Unix time
- `{duration}` — duration of the DVR segment in seconds
- `{path}` — path to the MP4 file on the server hard drive or Amazon S3 storage

are specified in the query string.

Export DVR segment to MP4 file on the server hard drive

To export a DVR segment to an MP4 file and upload it directly to the server hard drive, use the following command in the terminal:

```
curl -u username:password -X POST "http://FLUSSONIC-IP/streamer/api/v3/streams/STREAM_NAME/dvr/export?from=1675159800&duration=4200&path=/home/example/file.mp4"
```

Here:

- `from=1675159800` — start time of the DVR segment in Unix time
- `duration=4200` — duration of the DVR segment in seconds

> **Note**
>
> If `duration` is more than 10800 seconds (3 hours), file generation may take several minutes because *Flussonic* have to collect all the required archive fragments from the disks.

- `path=/home/example/file.mp4` — path to the MP4 file on the server hard drive

The recording will be exported as `file.mp4` and uploaded directly to the `/home/example` directory on the server hard drive.

> **Warning**
>
> Be careful not to overwrite the existing files.

Export DVR segment to MP4 file to Amazon S3

To export a DVR segment to an MP4 file and upload it directly to the Amazon S3 bucket, use the command below:

> **Warning**
>
> You should *URL encode* the query parameters when creating a URL to export a DVR segment to the Amazon S3 storage. **URL encoding** converts the query parameters so that they can be processed correctly when transmitted over the Internet.

```
curl -u username:password -X POST "http://FLUSSONIC-IP/streamer/api/v3/streams/STREAM_NAME/dvr/export?
from%3D1675159800%26duration%3D4200%26path%3Ds3%3A%2F%2FAWS_ACCESS_ID%3AAWS_SECRET_KEY%40s3.amazonaws.com%2Fbucket%2Fpath%2Fto%2Ffile.mp4"
```

Here:

- `from=1675159800` — start time of the DVR segment in Unix time
- `duration=4200` — duration of the DVR segment in seconds

> **Note**
>
> If you specify rather large value for `{duration}`, for example greater than 10800 seconds (3 hours), it may take several minutes for the download to start. This is because *Flussonic* have to prepare the file for export by collecting all the required archive fragments from the disks. This process may take a long time when there are a lot of fragments.

- `path=s3://AWS_ACCESS_ID:AWS_SECRET_KEY@s3.amazonaws.com/bucket/path/to/file.mp4` — path to the MP4 file in the Amazon S3 bucket with security credentials: `AWS_ACCESS_ID` access key ID and `AWS_SECRET_KEY` secret access key.

You can also save the metadata with the MP4 file by adding the `meta=true` to the query string:

```
curl -u username:password -X POST "http://FLUSSONIC-IP/streamer/api/v3/streams/STREAM_NAME/dvr/export?
from%3D1675159800%26duration%3D4200%26path%3Ds3%3A%2F%2FAWS_ACCESS_ID%3AAWS_SECRET_KEY%40s3.amazonaws.com%2Fbucket%2Fpath%2Fto%2Ffile.mp4%26meta%3Dtrue"
```

The metadata will be stored in the `udta.meta.ilst.data` atom.

Information and files available for end users to request

Requests available for end users can be authorized and unauthorized. We recommend always authorizing the requests to make access to the data secure. You can use the token-based authorization. In the examples below we will not be authorizing the requests.

REQUESTING JPEG THUMBNAILS

If you have configured thumbnails on your DVR or HTTP thumbnails, *Flussonic* writes the thumbnails on the disk. You need a thumbnail URL to access a thumbnail. These URLs contain the UTC time of the time frame in the video.

If you do not know the exact time frame to request a thumbnail, you can specify an approximate time frame. For example, you might know from the motion detector that something happened around some point in time, or you might know the necessary moment in time from the timeline of a player.

If no thumbnail at the specified time frame is found, *Flussonic* returns the thumbnail at the time frame closest to the specified one. *Flussonic* redirects the request and returns a ready-to-use thumbnail URL.

For example, we request a thumbnail for 2023/01/31 at 1:28:11 PM. No thumbnail is found at this time, but there is one close to this time frame. *Flussonic* returns `Location` with the time frame 2023/01/31 1:27:07 PM at which the thumbnail was found. This time frame can be used to get the necessary thumbnail.

To request the JPEG thumbnail of a stream from DVR, use the `Streaming-API: GET /{name}/{from}.jpg` call, where

- `{name}` — stream name
- `{from}` — Unix timestamp after which *Flussonic* searches for the written JPEG or the closest keyframe to generate a JPEG file from

```
curl -v 'http://FLUSSONIC-IP/STREAM_NAME/1675171691.jpg'
...
< HTTP/1.1 302 Found
< Connection: keep-alive
< Date: Tue, 31 Jan 2023 13:28:11 GMT
...
< Location: /STREAM_NAME/1675171627.jpg
```

GENERATING JPEG THUMBNAILS ON DEMAND

Flussonic can generate JPEG on-the-fly. It reduces disk space and disk I/O, but be careful and protect it with authorization because it is not cheap for the CPU.

To get the JPEG thumbnail, use the `Streaming-API: GET /{name}/{from}-preview.jpg` call, where

- `{name}` — stream name
- `{from}` — Unix timestamp after which *Flussonic* searches for the recorded JPEG or the closest keyframe to generate a JPEG file from

```
curl -v 'http://192.168.2.3:80/STREAM_NAME/1675179644-preview.jpg'
...
< HTTP/1.1 200 OK
...
< Content-Length: 5738
< Content-Type: image/jpeg
< Last-Modified: Wed, 02-May-2018 07:00:40 GMT
< X-Thumbnail-Utc: 1525244440
...here goes jpeg...
```

REQUESTING MP4 THUMBNAILS

We recommend that you stop using JPEG screenshots and use our video thumbnails. Video thumbnails of DVR are accessible in a similar way as JPEG screenshots. *Flussonic* corrects your request if no suitable frame is available at the specified time.

To request an MP4 thumbnail for the DVR recorded stream, use the `Streaming-API: GET /{name}/{from}-preview.mp4` call, where

- `{name}` — stream name
- `{from}` — Unix timestamp after which *Flussonic* searches for the recorded MP4 or the closest keyframe to return an MP4 file from

```
curl -v 'http://FLUSSONIC-IP/STREAM_NAME/1675252800-preview.mp4'
...
< HTTP/1.1 302 Found
< Location: /STREAM_NAME/1675252803-preview.mp4
```

You will receive an MP4 file consisting of one frame.

DOWNLOAD THE DVR SEGMENT TO MP4 OR MPEG-TS FILE TO A LOCAL COMPUTER

Flussonic allows you to download a DVR segment to MPEG-TS or MP4 file to your local computer.

To download a DVR segment to MP4 file, use the `Streaming-API: GET {name}/archive-{from}-{duration}.mp4` call, where

- `{name}` — stream name
- `{from}` — start time of the DVR segment in Unix time
- `{duration}` — duration of the DVR segment in seconds

Here are the URLs to download a one-hour DVR segment of a stream to an:

- MP4 file:

```
http://FLUSSONIC-IP/STREAM_NAME/archive-1525186456-3600.mp4
```

- MPEG-TS file:

```
http://FLUSSONIC-IP/STREAM_NAME/archive-1525186456-3600.ts
```

Opening such a URL in a web browser prompts a file to download on your local computer.

If you need to download a DVR segment with certain tracks only, specify them in the `filter.tracks` parameter in the query string:

```
http://FLUSSONIC-IP/STREAMNAME/archive-1525186456-4200.mp4?filter.tracks=v1a1
```

3.5.4 Keyframe-only export

Exporting keyframes to MP4 files

Flussonic offers an experimental feature of exporting keyframes only as MP4 files. It is useful in timelapse video creation.

To download MP4 files to a computer on the client side, use the following address:

```
http://FLUSSONIC-IP/STREAMNAME/archive-1350274200-4200.mp4?timelapse
```

a request for a keyframes-only file at 25 fps.

```
http://FLUSSONIC-IP/STREAMNAME/archive-1350274200-4200.mp4?timelapse=20
```

a request with fps correction. The exported file will have a 20-second length.

3.5.5 Flussonic RAID for DVR

Flussonic RAID for DVR is an application-level RAID offering high reliability and convenience when writing video data to dozens of disks.

Flussonic RAID has substantial advantages over similar solutions:

- No need to buy expensive RAID controller hardware for 60 disks, for example. You can use all disks in JBOD mode (Just a Bunch of Disks). Different disk sizes are allowed. You format each disk separately and mount them in the system in a specific directory. After that, you set up Flussonic and it begins to monitor the condition of the disks and allocate the data among them. You can manage data allocation by using configuration options described later on this page.
- Reliability: If any drive fails, data will continue to be recorded to other drives. Only the part of the archive that was written to a failed drive can be lost.
- Continuous operation: You can add and remove disks from the array while Flussonic is writing to the archive. The changes are applied without restarting Flussonic.
- Automatic seamless migration of data between RAID disks, which makes it possible to empty a disk while keeping the DVR archive readable and accessible.
- Automatic data allocation between disks in an array: Flussonic decides on which disk it would be better to write. The amount of data can be bigger than it's possible to write to one disk within acceptable time limits, so Flussonic uses even data allocation among disks. To minimize electricity costs, you can limit the number of disks that can be written on simultaneously.
- Protection from writing if disks were not mounted. This will prevent all the data from being written to the root partition.

On this page:

Setting up application-level disk array in Flussonic Media Server

Existing archives cannot be transferred to RAID, you can only start writing new archives to it. To start working with RAID, you'll need to configure the server by adding disk array settings (these are essentially global DVR settings) and specify this array in a stream settings to record the stream into this array.

The setup procedure is different for Flussonic Watcher and for Flussonic Media Server.

The DVR disk array is created at the operating system level when you mount the disks, and then the entire array is managed programmatically by the Flussonic server.

Disk array settings act as global DVR settings. Flussonic allows you to specify array settings in the configuration file `/etc/flussonic/flussonic.conf` or in the web UI.

Configure DVR settings in the following order. First, specify the disk array settings, for example:

```
dvr my_raid {
  root /storage/raid;
  raid 0;
  metadata idx;
  disk volume1;
  disk volume2 keep;
  disk volume3 migrate;
}
```

Then specify that a stream must be recorded to the disk array with the archive depth of seven days:

```
stream channel5 {
  input fake://fake;
```

```
  dvr @my_raid 7d;
}
```

This stream receives the global settings specified in `dvr my_raid`. You will have the opportunity to override some of them in the stream settings.

RAID CONFIGURATION EXAMPLE WITH THE FULL SET OF OPTIONS

Disk array has three kinds of settings:

- Global DVR disk array settings
- Settings that can be overridden in individual stream settings
- Options that control the recording process on disks.

Below is the description of all settings.

Global DVR disk array settings, which apply only to the DVR on a disk array:

- **`root`** — the base directory where disks are mounted and indexes are located. Example: `root /dvr/raid;`
- **`raid`** — enables the work with the array (the allowed value is 0). If you enable RAID, it checks for active disks. Flussonic will check the major device of the root path and files in directories and will not allow writing if a directory was not mounted successfully. Example: `raid 0;`
- **`active`** — the number of disks to which data will be written. With a large number of disks, to record all at once would be uneconomical (high power costs), so you'd better write only to a few disks simultaneously. If you do not specify this option, all disks with sufficient free space will be written to. Example: `active 2;`
- **`metadata`** — the subdirectory in `root` for storing cached metadata. *You don't need to create this directory manually*, it will be created when the configuration is applied. We recommend using SSD for quick access to the archive. Example: `metadata idx;`
- **`disk`** — the path to a mounted disk. The paths specified in the `disk` option must be real mount points. For example:

```
Filesystem                        Size  Used Avail Use% Mounted on
/dev/mapper/pve-vm--15--disk--1  7.9G  5.7G  1.8G  77% /
/dev/loop0                       196G  4.0G  182G   3% /dvr/_raid_/d1
/dev/loop1                       196G   4.0G  182G   3% /dvr/_raid_/d2
```

Settings that apply to individual streams (when specified in global DVR settings, they will apply by default to all streams, but you can override them in the settings of a single stream):

- **limits** — sets limits on the size and depth of the archive. Example: `90% 3G 1d`.
- **replicate** — sets the replication of a DVR archive. Port is optional. Example: `replicate port=2345;`
- **copy** — copies the data in parts to another location. Example: `copy /opt/storage;`
- **schedule** — sets the schedule for recording video to archive. Example: `schedule 3-6,5-8,23-5;`

Find the modes for managing the recording process in the API Reference.

Mounting disks in Linux

Flussonic RAID is a software RAID, meaning that disks should be mounted as general `ext4` separate disks. You are not limited to using the `ext4` filesystem but we strongly recommend it if you have no weighty reasons to use another filesystem.

Here is the real configuration of one of our laboratory servers:

```
root@dvr:~# lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0   9.8T  0 disk /storage/d1
sdb      8:16   0   9.8T  0 disk /storage/d2
sdc      8:32   0   9.8T  0 disk /storage/d3
sdd      8:48   0   9.8T  0 disk /storage/d4
sde      8:64   0 119.2G  0 disk
└─sde1   8:65   0 119.2G  0 part /

root@dvr:~# cat /etc/fstab
# / was on /dev/sda1 during installation
# OS SSD DISK
UUID=5081dd01-6b97-4166-b05c-e9f59476b553 /               ext4    errors=remount-ro 0       1

#sda
UUID=8c5bcc39-8599-4545-a373-f63a441b53b8 /storage/d1    ext4    defaults,nofail,x-systemd.device-timeout=5  0   2
```

```
#sdb
UUID=f4888c12-6faa-4ac1-b4fb-e04c3e4ddc31 /storage/d2   ext4    defaults,nofail,x-systemd.device-timeout=5      0       2
#sdc
UUID=7feedd26-feb4-47ad-99b8-ec732cbd87aa /storage/d3   ext4    defaults,nofail,x-systemd.device-timeout=5      0       2
#sdd
UUID=ed41fad5-92fd-4737-87ff-c6a859aeed10 /storage/d4   ext4    defaults,nofail,x-systemd.device-timeout=5      0       2
```

Key points:

- OS Disk must be an SSD disk.
- All HDDs for Flussonic array must be mounted to the same root directory.

Flussonic configuration:

```
dvr raid {
  root /storage;
  raid 0;
  disk d1;
  disk d2;
  disk d3;
  disk d4;
}
```

Please consult with our local system administrator or our support team if you have any questions about Linux mount or Flussonic RAID configuration.

Reading runtime statistics

The runtime statistics about RAID is now included in the response to the API call `dvr_get`, Learn more about the call in HTTP v3 API.

The statistics shows the state of disks in the RAID:

- `status` — whether the disk is mounted or not
- `blobs_count` — the number of blobs on the disk
- `size` — the size in bytes
- `usage` — disk utilization percentage
- `io_usage` — disk utilization percentage from /proc/devstat.

If migration is taking place, the response shows the speed of migration, the estimated end time, and the time when values were last changed:

- `migration_speed` — the speed of copying the last blob, in bytes per second
- `migration_eta` — the estimated migration end time, in UTC seconds
- `migration_updated` — the time when the values of `migration_speed` and `migration_eta` were last updated.

After the migration has been completed, this parameters take the value `undefined`.

Global DVR settings in the web UI

This section describes how to write the archive to Flussonic's apllication-level RAID.

To add global DVR settings including RAID arrays:

Go to **Config** > **DVR** > **Add DVR** and define the RAID settings. To add more than one array, use the **Add DVR** button.

Copy chunks to this location is used for copying the archives of static streams to the specified path.

> **Note**
>
> Another way to open global DVR settings is from the settings of (any) stream in the stream's **DVR** section by clicking the **Edit DVR configurations** link under **Global DVR config**.

To apply global settings to a stream and override some of them:

In the settings of this stream in the **DVR** section, click in the **Global DVR config** field and select the previously created array in the list that appears. In the example, an array `my_raid` was chosen:

Fields on the **DVR** page will be filled with values from the selected global configuration (Path, Copy chunks to this location, and so on), and you are free to change them for this stream, if necessary.

The stream's archive will be recorded to the RAID array that you have chosen.

To open global DVR settings, in the stream's **DVR** section click the **Edit DVR configurations** link under **Global DVR config**.

How to replace a failed disk in Flussonic RAID

Replacing a failed disk in Flussonic RAID generally consists of the following steps:

- Enable for the disk the "Abandon" mode in the tab "DVR" for the streamer. The description of disk modes can be found in the documentation here:

https://flussonic.com/doc/api/reference/#tag/dvr/operation/dvr_disk_save|body|mode

- At the beginning of the next hour, check that writing to this disk is no longer performed:

```
"lsof -p pidof streamer | grep 'devicename'"
```

where `devicename` is the name of the disk in /dev/.

- Replace the "failed" disk with a new one, and check its name - it should not change when saving the mount point on the server.
- Partition and format the new disk, mount it at the same path as the old disk - check the settings in /etc/fstab and the "dvr watcher" section in "/etc/flussonic/flussonic.conf" or in the "DVR" tab for the streamer in Watcher UI.
- Switch the mode for this new disk in the Flussonic raid from "Abandon" to "Normal" in the tab "DVR" for the streamer.
- At the beginning of the next hour Flussonic will start writing the archive to this new disk.

> **Warning**
>
> If the server does not support hot swap disks, then at the step 3 replace the failed disk when the server is shut down.

3.5.6 Working with DVR Archive in Middleware

This article aims to assist Middleware developers in implementing video archive functionality to fulfill various user scenarios, including but not limited to:

- Pausing live broadcast to take a break
- Watching the current program from the beginning because they like the movie and want to start over
- Watching yesterday's program
- Bookmarking a favorite program to watch again in a few months

The problem addressed in this article is that the only protocol technically capable of easily implementing those scenarios is RTSP. However, RTSP has long been abandoned in favor of segment-based HTTP protocols: HLS, DASH (and a bit of MSS). None of these mainstream protocols for IPTV OTT support archive and DVR functionality out-of-the-box, hence the need for technical ingenuity to solve these tasks.

How to Best Manage Access to the Archive?

The most effective way to facilitate access to the archive is by using epg-vod for playing archived content, and using event for playing live broadcasts.

Playing Archive by EPG

For this method to work, the Middleware must maintain a precise EPG schedule. References to timestamps and time will be made throughout this section. We only consider seconds in UTC (Epoch time). Under no circumstances should local time be used, as it introduces unnecessary confusion.

When watching a program, it is recommended to store the current viewing time in the middleware database, so that upon reopening the user can choose whether to resume or to start from the beginning.

For example, when a user selects the program that aired from 1717677139 UTC to 1717679255 UTC from the archive, the following URL must be generated for the player: `http://FLUSSONC-IP/STREAM_NAME/archive-1717677139-2116.m3u8?event=true`

This URL allows the player to handle:

- **Seek within the program**
- **Pause**
- **Fast forward playback**

At the end of playback, it is recommended to switch to the next program in the EPG to ensure a continuous flow of content to the viewer.

Viewing the Current Broadcast with Event Playlists

Viewing the current broadcast is technically more challenging. We recommend using event playlists, but they do not allow rewinding in native Safari, making them unsuitable for TV. In such cases you would need to write your own timeline code in JavaScript. If Safari is not critical for the operation, you can confidently use the same URLs as in epg-vod:

`http://FLUSSONIC-IP/STREAM_NAME/archive-1717677139-2116.m3u8?event=true`

The trick is that if the playlist's closing moment is in the future, it will be served not as VOD, but as EVENT, allowing the player to re-request it and continue playing.

After this playlist ends, you should also switch to the next one according to the EPG and continue viewing.

Using such a URL allows pausing the live stream and resuming from the same spot. It is important to note that you need to program the "live" button yourself, because during the pause the playlist might automatically switch from EVENT to VOD and close. In this case, you need to determine the next program in the EPG and jump to it.

3.5.7 How to save nPVR recordings

One of the most important differences between an IPTV OTT service and linear TV is access to the broadcast archive. This article describes how to ensure the preservation of TV programs that a customer has set aside for long-term storage. At the same time, the remaining unnecessary part of the archive should be deleted to free up disk space.

Until 2023, we offered a mechanism called DVR locks to solve this problem, but it was abandoned because it scales very poorly and is even worse for backup.

This document does not address the task of ensuring transparent access to TV broadcasts when transferring the archive from hot storage to cold storage as individual video files. This extended task is handled by Flussonic Central, which is recommended for use. Here, the basic method with just Media Server is described.

Regulatory and Legal Considerations

In some countries, recording the archive by the operator may either be prohibited or require separate agreement/regulation.

Meanwhile, recording a broadcast to a set-top box for personal use by subscribers is generally permitted. Recording to a set-top box constitutes a significant increase in the service cost, as it requires selling a more expensive device to each subscriber, which includes an additional component that can wear out.

nPVR (network Personal Video Recorder) provides a good solution to this issue: a subscriber orders the recording of a broadcast, and it is saved on the server and stored upon their request.

Solution Scheme

- The client's server must receive EPG and save the TV programs in the database as separate records.
- Based on subscribers' preferences, the client's server should mark the demanded necessary TV programs as protected from deletion.
- The Media Server will regularly request a list of episodes from the client's server for each stream that needs to be kept in the archive.
- If this list of episodes is given as TV programs, they will be saved in the archive as long as there is enough disk space.

digraph { EPG -> Portal [label = "fetch episodes"] Customer -> Portal [label = "select for recording"] Portal -> Flussonic [label = "config_external"] }

Interaction between Media Server and Client's Server

To activate the episode polling mechanism, the client's server must:

- For the operation of episodes, Media Server should use the stream management system config_external;
- In the response with the stream list, it is necessary to include the X-Config-Server-Episodes header. Upon seeing this header, the media server will enable episode polling.
- Implement the method for providing the episode list.

In the `external_episodes_list` method, it is very important to pay attention to the `media` parameter with the list of requested streams. The absence of data for a stream in the response means the deletion of the entire archive for that stream. That is, returning an empty list means deleting the archive for all streams listed in the `media` parameter.

In this response, it is permissible to return a 400 or 500 code if, for some reason, there is no ready response with the list of episodes.

How will it work?

- Media Server is configured with a stream archive depth of, for example, 3 days of continuous archive.
- All settings are received through `config_external` from the server that sends the signaling header `X-Config-Server-Episodes: true`.
- Based on this, periodically, Media Server makes a request to `external_episodes_list`, polling the ranges scheduled for cleaning.
- Those archive fragments that are covered by episodes remain in place, while the rest are deleted.

Exporting Recordings as Files

In cases where long-term storage with infrequent access is required, it makes sense to export archive fragments as files to a separate external storage.

This task is handled by Flussonic Central.

3.5.8 How to delay TV playback in another time zone

Many TV channel broadcasts are intended for only one time zone.

If you want to distribute the same channel to users in Germany or in the USA, you will face a problem: people have an early morning, but they are already watching evening broadcasts.

Flussonic can delay stream playback for a few hours, so that people in a different time zone watch the "Good morning" broadcast in the morning, and not late at night.

There are several technical ways to organize this in Flussonic Media Server, based on the frequency of addressing various channels in different time zones.
The difference between these methods is the number of times that the archive is read for delayed playback of the channel.
You can start playing the delayed stream, and the archive will be read once, regardless of the number of people willing to watch it, or you can provide personalized URLs to the users, and the archive will be read for each user individually.

If about 250 channels are written, and you wish to broadcast to 3 locations, you will get a total of 250 channels to write, and 750 to read. It makes sense to leave some channels constantly running, and start some channels only at the request of users.

Delayed stream

Assume we have a configured channel:

```
stream channel {
  input fake://fake;
  dvr /storage 1d;
}
```

The channel must have a configured archive (`dvr /storage 1d` in the example). Now we can create a second stream:

```
stream channel-1hour {
  input timeshift://channel/3600;
}
```

This stream will read from the archive and play the video with a one-hour (3,600 seconds) delay.

You can create as many streams as you wish.

Personal access to the archive

If you have a configured stream:

```
stream example_stream {
  input udp://239.1.2.3:1234;
  dvr /storage 1d;
}
```

it can be assigned URL:

- for playback over HTTP MPEG-TS:

```
http://FLUSSONIC-IP/example_stream/timeshift_rel/3600
```

- for playback over HLS:

```
http://FLUSSONIC-IP/example_stream/timeshift_rel-3600.m3u8
```

Multilingual channels can be assigned for set-top boxes:

```
http://FLUSSONIC-IP/example_stream/timeshift_rel_video-3600.m3u8
```

In this case, each client will individually read the archive. This method should be used for rarely used combinations of a channel and a time zone.

Skipping gaps in timeshift playlist

If you have gaps in your archive (e.g. if your source was down for couple of minutes), then at reaching that gap Flussonic Media Server will return empty playlist while playing HLS timeshift.

If it's acceptable to break the time shift and skip this gap, you may specify playlist URL with the `ignore_gaps=true` parameter:

```
https://FLUSSONIC-IP/STREAM_NAME/timeshift_abs-123123123.m3u8?ignore_gaps=true
```

timeshift_abs HLS URLs present a great difficulty caused by the nature of the HLS Protocol. The fact is that Flussonic can only probabilistically join separate HTTP requests into the same session. Flussonic believes that the session is the same, if for two queries, client IP address, channel name, query protocol and the token match. In case of several consecutive timeshift_abs requests, Flussonic will decide that it's the same session, in the end, it may distort viewing. To avoid this, a new token should be passed in the timeshift_abs request.

A simpler variant is requesting an HTTP-MPEGTS `http://FLUSSONIC-IP/STREAM_NAME/timeshift_abs-1429829884.ts`. However, the HTTP MPEGTS option denies access to multi-bitrate.

3.5.9 How to set up a redundant DVR service

When building an IPTV service, you need to ensure that:

- The DVR archive is preserved.
- The service is always available.

To achieve these goals, set up a redundant DVR service. To do that, use the Flussonic *cross-replication* mechanism.

On the page:

- What's cross-replication and why do you need it
- How cross-replication works
- How to configure cross-replication

What's cross-replication and why do you need it

Cross-replication is an archive redundancy mechanism where two Flussonic servers record and store the stream archive. These servers can access the source and restore missing segments of the archives from each other (for details see How cross-replication works). The archives on the servers are full copies of each other.
If one of the servers becomes unavailable, the second one continues to record the archive, accessing the source directly. When the first server comes back online, it automatically retrieves the missing segments from another server.

Cross-replication is replication from the first server to the second one and vice versa.

With cross-replication, your service:

- Continues providing the archive recordings to viewers if one of the servers with the archive crashes or becomes temporary unavailable.
- Restores missing segments of the archive when the server returns online by transferring data from the working server.
- Ensures that the archives on the servers are identical.

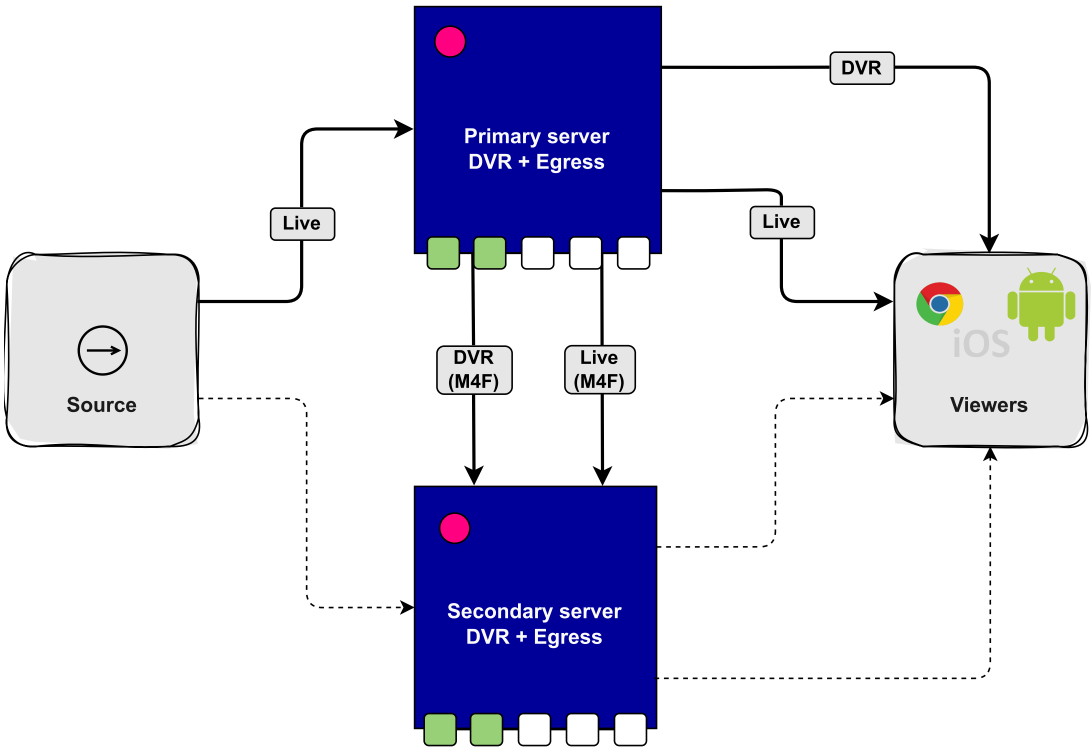
How cross-replication works

> **Note**
>
> When transmitting streams between Flussonic servers, it's recommended to use Flussonic's proprietary protocol—M4F. Read more about the benefits of the M4F streaming protocol in the M4F protocol overview section.

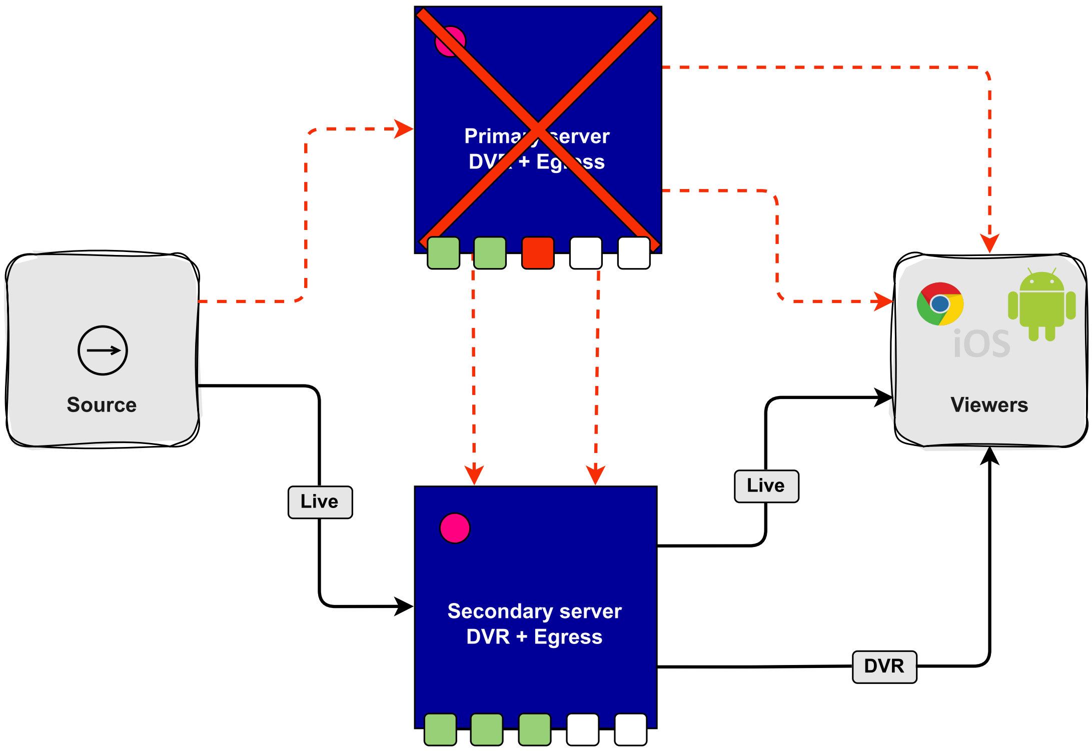
Cross-replication mechanism in Flussonic applies to a stream archive, rather than a server archive. If you need to configure cross-replication to several streams, you should configure it for each stream.

Cross-replication works in three modes:

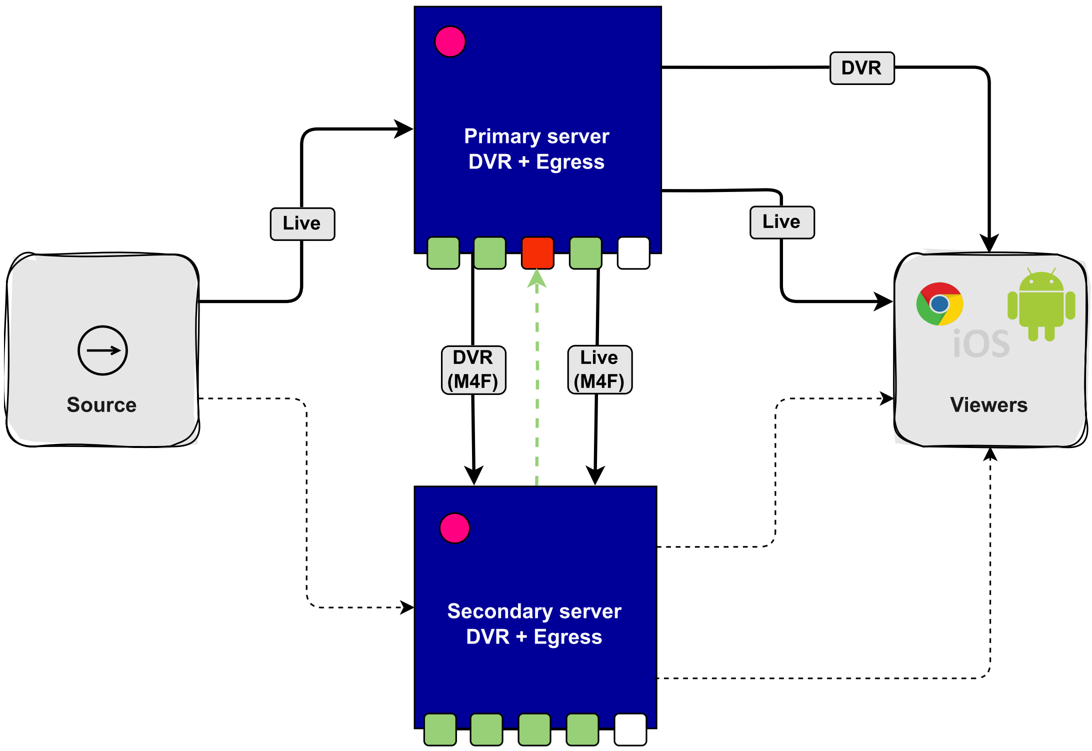
- Normal mode:



- The primary server captures the live stream from the UDP source and writes it to the archive of some `example_stream`.
- The secondary server captures the live stream and the archive of the primary `example_stream` stream from the primary server via M4F to the backup stream `replica_example_stream`.
- Emergency mode:



- The primary server goes offline, stops receiving the live stream from the UDP source and writing it to the archive.
- The backup `replica_example_stream` stream switches directly to the UDP source. The secondary server receives the live stream from the UDP source and keeps writing the archive of the stream.
- Disaster recovery mode:



- The primary server comes back online and the primary stream `example_stream` switches back to the UDP source. The server captures the live stream from the UDP source and writes it to the archive.
- The primary server retrieves the missing segments of the archive from the secondary server while the primary server was offline.
- The backup stream switches back to the primary server. The secondary server captures the live stream from the primary server and keeps writing the stream to the archive.

How to configure cross-replication

To set up cross-replication of a stream `example_stream` on two Flussonic servers, configure the following on both servers:

- ingest from the source (`input udp://`)
- ingest from the Flussonic server for replication (`input m4f://`)
- DVR on both servers (`dvr /storage 3d`) restreaming DVR `remotes=m4f://` and replication with the `replicate` option

Suppose `primary_flussonic.example.com` is the primary server, and `secondary_flussonic.example.com` is the secondary server.

Primary stream configuration on the primary `primary_flussonic.example.com` server:

```
stream example_stream {
  input udp://224.1.2.3:1234;
  dvr /storage remotes=m4f://secondary_flussonic.example.com/replica_example_stream 3d replicate;
}
```

Backup stream configuration on the secondary `secondary_flussonic.example.com` server:

```
stream replica_example_stream {
  input m4f://primary_flussonic.example.com/example_stream;
  input udp://224.1.2.3:1234;
  dvr /storage remotes=m4f://primary_flussonic.example.com/example_stream 3d replicate;
}
```

3.5.10 How to quickly copy the DVR archive to a secondary server and increase the simultaneous number of viewers

On this page:

Replication is a mechanism for copying DVR (Digital Video Recorder) archive from the primary server to secondary ones. You can automatically recover the lost archive segments after a crash or a failure.

Why use DVR replication?

You can use the *Flussonic* replication mechanism to:

- Extend the DVR service and raise the simultaneous number of viewers.
- Quickly start a new DVR server or a DVR server after a crash.
- Automatically recover a DVR archive after a crash.

90% of the time the primary DVR server is enough to serve all the viewers. The primary server writes the archive to the disk and delivers the segments to the viewers from any point of the timeline in the DVR archive. This way, viewers can watch recorded live streams at different times, with one viewing 24 hours after the live stream and another viewing 72 hours after, each receiving unique content.
In the evening, when people return home from work, the number of viewers increases.
To serve more viewers and deliver archive segments without overloading the primary server:

1. Connect secondary server.
2. Replicate the archive from the primary server to the secondary server.
3. Redirect new viewers to the secondary server.

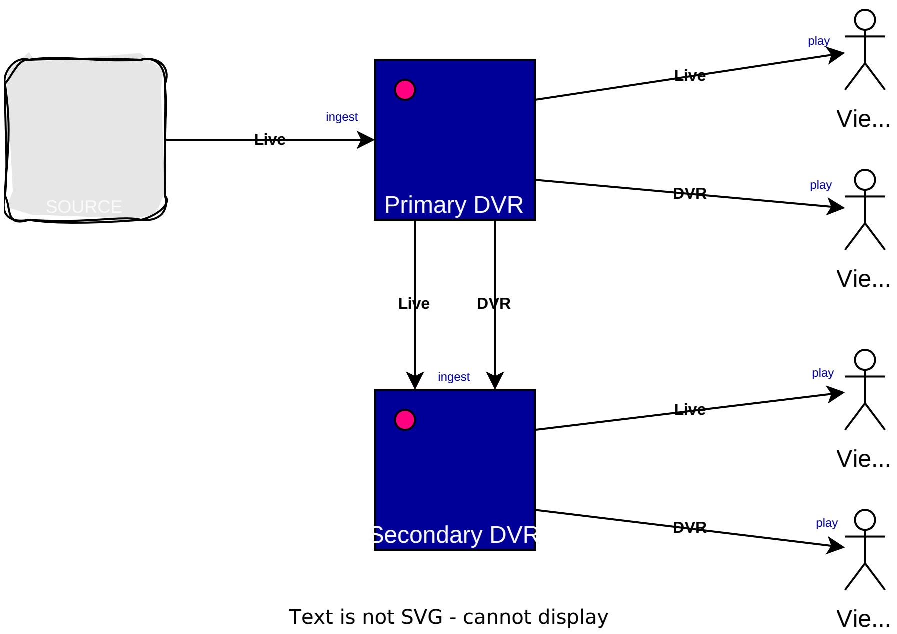
Based on the viewing statistics on the DVR content, the time between the first and the second request of the same archive segment is 18 hours. Hence, it's cheaper and easier to copy the archive to the secondary server than configuring local cache. This way, viewers can watch the content without interruptions.

By adding a secondary server the service bandwidth and the disk read speed double, allowing to serve twice as many viewers. The main goal of the primary server isn't to overload the replication channel to the secondary server while also distributing content to viewers.

When you need to get a secondary DVR server up and running in a short period of time to deliver archive segments to viewers, use replication. Instead of recording a week's worth of archive from scratch on a secondary server and waiting seven days, you can replicate the archive from the primary server in a few hours and have the secondary server ready. Segments of the archive are time-synchronized, so viewers won't see any difference in the content they receive.

How DVR replication works

Diagram 1. DVR replication process



Primary server:

1. receives a live stream from the source
2. records and stores it
3. broadcasts the live stream and recording segments to viewers

Primary server doesn't know about the secondary server. When the secondary server starts, it does the following:

1. Connects to the primary server.
2. Captures the live stream and copies the archive from the primary server. Because of this, the read speed from the primary server disk increases along with the write speed to the secondary server disk.
3. As the secondary server finishes replicating the archive, it delivers the archive segments to new viewers.

How to configure DVR replication

You can configure replication of all streams or a certain stream from the primary server.

DVR REPLICATION OF ALL STREAMS

You can set up DVR replication of all streams in the Flussonic UI or in the configuration file `/flussonic/flussonic.conf`.

In the Flussonic UI

To turn on the replication of all streams from the primary server to the secondary server in the Flussonic UI, follow these steps:

1. On the secondary server, in the drop-down menu on the left, open the **Config** section and go to the **DVR** tab.
2. In the **Additional** section, check the *Dvr replicate* box. Specify a separate port for replication in the *Replication port* field if required (see How to specify a separate port for DVR replication).

1. Save the settings by clicking the save icon in the upper-right corner.

After that the secondary server will start replicating the archive.

In the configuration file

To replicate all streams from the primary server to the secondary server, specify the `source` directive with the `replicate` option in its DVR settings in the secondary server configuration:

```
cluster_key abcd;
source primary-server {
  dvr /storage 20d replicate;
}
```

DVR REPLICATION OF A CERTAIN STREAM

You can set up DVR replication of a certain stream in the Flussonic UI or in the configuration file `/flussonic/flussonic.conf`.

In the Flussonic UI

To turn on the replication of a certain stream from the primary server to the secondary server in the Flussonic UI, follow these steps:

1. On the secondary server, create a new stream and specify the required stream from the primary server as input via M4F or M4S. Save the stream settings by clicking **Save**.

1. Open the **DVR** tab in the stream settings and specify the required archive depth. The secondary server will copy the archive from the primary server according to this value.
2. Check the *Dvr replicate* box. Specify a separate port for replication in the *Replication port* field if required (see How to specify a separate port for DVR replication).

1. Save the changes by clicking **Save**.

After that the secondary server will start replicating the archive.

In the configuration file

To configure replication of a certain stream from the primary server to the secondary server, in the configuration file `/flussonic/flussonic.conf` of the secondary server do the following:

1. Create a new stream and specify the required stream from the primary server as input via M4F or M4S: `input m4f://PRIMARY-SERVER-IP/STREAM_NAME`.
2. In the stream settings, specify the required archive depth. The secondary server will replicate the archive from the primary server, according to this value: `dvr /STORAGE_NAME 7d`.
3. In the DVR `dvr` settings of the stream, add the `replicate` option. Specify a separate port for replication in the `replication_port=` parameter if required (see How to specify a separate port for DVR replication).

See the examples of configurations of the primary and secondary servers:

- **Primary server stream configuration**:

```
stream fake {
  input fake://;
  dvr /storage 7d;
}
```

- **Secondary server stream configuration**:

```
stream repl_example1 {
  input m4f://primary-server-ip/fake;
  dvr /storage 7d replicate;
}
```

In the example, the primary server connects to the stream source, and the secondary server copies the archive from the primary server as opposed to the cross-replication.

Warning

Don't use the `dvr_offline` option, which turns off the recording, instead of `dvr` because the `replicate` option enables recording automatically.

HOW TO SPECIFY A SEPARATE PORT FOR DVR REPLICATION

Note

Replication works with the *Flussonic* protocol—M4F. Use M4F when transmitting video between *Flussonic* servers. You can read more about the benefits of the M4F protocol in the M4F overview.

By default, replication uses the port specified when configuring the M4F source. To prevent channel from overloading and your service from failing, you can specify a separate port for replication. You can do that in the Flussonic UI or in the coonfiguraiotn file `/flussonic/flussonic.conf`.

In the Flussonic UI

1. If you configure replication for a certain stream, in the stream settings of the secondary server, go to the **DVR** tab. Check the *DVR replicate* box and specify the replication port in the *Replication port* field.

   If you configure replication for all streams, on the secondary server, in the drop-down menu on the left, open **Config** and click the **DVR** tab. In the **Additional** section, check the *Dvr replicate* box and specify the replication port in the *Replication port* field.

   Click **Save** to save the settings.
2. On the primary server, in the drop-down menu on the left, open the **Config** section. On the **Settings** tab, under **Listeners** section, specify the same HTTP port as in the earlier step by clicking the add icon. Click **Save** to save the settings.

In the configuration file

1. In the configuration file of the secondary server, specify the port number in the `replication_port` parameter next to the `replicate` option in the stream settings, if you configure replication for a certain stream. If you configure replication for all streams, do the same, but in the `source` directive:

```
stream repl_example2 {
  input m4f://primary-server-ip/fake;
  dvr /storage 7d replicate replication_port=8002;
}
```

1. In the primary server settings, specify the same port number using the `http` option:

`http 8002`

See also: Cross replication for archive restoring.

3.5.11 DVR in a cloud

Storing archives in a cloud

Flussonic can record streams' archives to remote HTTP storages including Amazon S3 and OpenStack Storage (Swift).

Two methods of recording to cloud storage are supported:

- A stream is recorded directly to the storage segment by segment (by default). In this case, the archive in the cloud is always up to date, but this can be expensive if the provider charges for each recording operation. In addition, simple S3-compatible services (for example, MinIO) may not be able to cope with such a large number of files, while for a service with just a several dozen channels and several days of recording we are talking about millions of files.
- Recording in longer segments of one hour (use the **copy** option, see below). In this case, a local storage is required to accumulate recordings for the hour. This method reduces the operations of recording to the cloud and provides tolerance to short-term network failures because the hour-length fragment is transferred asynchronously.

Examples of configuration

To store a stream on **Amazon S3**, configure it like this:

```
stream chan1 {
  input fake://fake;
  dvr s3://minioadmin:minioadmin@minio:9001/test 10G;
}
```

To store a stream on **Amazon S3** and enable access **via HTTPS**, configure it like this:

```
stream chan5 {
  input fake://fake;
  dvr s3s://minioadmin:minioadmin@minio:9001/test 10G;
}
```

To store a stream in **OpenStack Storage (Swift)**, configure it like this:

```
stream chan2 {
  input copy://chan1;
  dvr swift://user=test:tester&password=testing@swift:8080/test 10G;
}
```

To store a stream in **Akamai Storage**, configure it like this:

```
stream chan3 {
  input copy://chan1;
  dvr akamai://keyName:keyValue@akamaihd.net/cpCode/dvr 10G;
}
```

Copying video archives to the cloud

The **copy** option helps significantly reduce the number of times that *Flussonic* accesses the disk on a cloud.

Flussonic first accumulates recorded video data on a local disk (in the specified directory). Then, once an hour, it moves the data to the cloud.

Specify the **copy** option like this:

```
stream chan4 {
  input copy://chan1;
  dvr /storage copy=s3://minioadmin:minioadmin@minio:9001/test 10G;
}
```

Recording to the network storage when a stream was migrated

The group of *Flussonic* servers can work with the same storage, keeping all records in one directory. When a stream migrates from one server to another, the new server will catch the recording made by the old server.

Flussonic completely transfers the configuration of the stream to the new server, and the archive will continue to work automatically.

Warning

Multiple servers must not record the same stream at the same time.

3.5.12 Recording DVR Archive on NAS NFS

Deploying virtualized corporate environments often involves the use of network storage as a reliable way to save a virtual machine image and run it on another server. In this setup, a single network storage can be connected to dozens or even hundreds of virtual machines.

This means that network storage is used in configurations where the computing power is significantly greater than the data, and the traffic from each computing node is relatively small.

Sometimes, this experience is attempted to be applied to video archive storage, raising the question: how to use network storage in video surveillance or television tasks.

The simple answer is: not at all. It is economically and technically unjustifiable, leading to enormous cost overruns and guaranteed issues with recording stability over NFS.

Network storage is essentially a server with specialized software for storage and arbitrary recording. Often, the size of the network storage is smaller than the space on a server with 36 disks assembled for video surveillance.

As a result, you are simply buying two servers, connecting them over a network, and copying from one to the other. No consolidation effect is achieved by connecting 10 video servers to one storage.

Storage Backup

Network storage typically uses some form of RAID with protection against one or two disk failures. Network storage resilient to the failure of an entire server is very rare.

If you need reliable archive storage, it is better not to rely on a single point of failure, but rather build a redundant cluster.

How to Connect Storage, If You Really Want To?

Network storage is usually mounted via the NFS protocol. Flussonic does not have built-in NFS client support, instead, it is expected to use the standard Linux client.

You should be aware that if any packets in the NFS traffic are lost, the streamer itself can freeze and become unresponsive indefinitely, requiring a server reboot due to NFS implementation specifics in the kernel.

Options like `soft`, `intr`, `timeo`, `retrans` can help. For more detailed explanations, refer to the operating system manuals: `man nfs`.

3.6 VOD

3.6.1 VOD files

VOD (Video On Demand) service is an integral part of services based on video delivery. It is a media delivery system, allowing users to access the content at any time regardless of the usual TV broadcasting schedule. VOD has a wide application field, for instance, education.

Flussonic Media Server supports playing video files on client devices and apps. A *virtual filepath*, called a ***VOD location***, must be set up to enable this feature. One VOD location can contain multiple directories. Multiple VOD locations can be used to arrange video files and apply different sets of settings to files for each VOD location.

Supported containers and codecs

The VOD broadcasting feature only supports playback of video files in MP4 containers (popular file extensions include `.mp4`, `.f4v`, `.mov`, `.m4v`, `.mp4a`, `.3gp`, and `.3g2`.) The H.264, HEVC video codecs are supported. The AAC, MP3, AC3, PCMA, and PCMU audio codecs are supported.

> **Warning**
>
> We strongly recommend that you convert files from MKV into MP4 because the MP4 format is much better for playing files via HLS or DASH. You can use ffmpeg to convert video files from MKV to MP4.

Containers	Video Codecs	Audio Codecs
MP4 (.mp4, .f4v, .mov, .m4v, .mp4a, .3gp, .3g2)	H.264, H.265	MP3, AAC (all profiles)

As you can see from the list, *Flussonic* does not support the MKV format, and there are reasons for this.

In an MP4 file, the header contains all data about tracks and segments in advance. It is enough to read the moov structure of an MP4 file for *Flussonic* to find out everything about all the frames (except their contents). And since moov takes up less than 1% of all data, Flussonic only needs to read a very small part of a multi-megabyte file. And this data is enough to create an HLS or DASH playlist.

The most important thing here is that moov contains bitrate data, so in the case of MP4, the player will immediately get a valid master playlist with track bitrate data, which will allow playing the file without errors. If there is no bitrate data, the player will not be able to select a track to be played. There may be other errors that are not easily fixed.

In the case of MKV files, the data about file structure might be missing. MKV packers sometimes specify NUMBER_OF_BYTES, but not always, and in this case Flussonic would have to read an entire file when opening it, in order to find out its contents and create a playlist.

Fragmented MP4

Flussonic supports VOD files in fragmented MP4 (fMP4) format. Fragmented MP4 does not have its own file extension and uses the same .mp4 extension as standard MP4 files. The difference between fMP4 and traditional MP4 is that:

- moov describes tracks with decoder settings, but does not describe frames.
- Data is divided into small pieces named fragments (same as segments in HLS, DASH and Flussonic).
- Each fragment is described by a moof (movie fragment) and mdat region which can be decoded without having other fragments (only the moov from the beginning is needed).
- At the end of the file there is an mfra index to help the player draw the timeline and navigate through it.

Compatibility with fragmented MP4 is important for VOD because many popular programs such as OBS Studio record video in this format. You can simply place the recording made by such a program in the VOD directory and distribute it from Flussonic Media Server. No additional settings is needed.

3.6.2 How to view a file?

How to view a file

The task: you have a video file, and you need to organize broadcasting of this file over a network.

Content:

Installing Flussonic Media Server

First, you should install Flussonic Media Server.

Preparing the file: Correct format

Flussonic Media Server can only play files in certain formats.
The main container format is mp4, video codec is h264, and audio codec is aac.

In order to be able to check and change the file format on the server, if necessary, you have to install ffmpeg.
It is not necessary to install it on the server itself, it can be installed on another computer with Linux, Windows or OSX operating systems.

Instructions for downloading ffmpeg are available from the official website: https://www.ffmpeg.org/download.html.

Many GNU/Linux distributions already have ffmpeg in their standard repositories. For example, in Ubuntu starting with version 15.04 Vivid Vervet, it is enough to enter "apt-get install ffmpeg" in the command line. If you don't have this package, you should search in third-party repositories, or just download static build.

To detect the format, we will use command ffprobe in the just installed ffmpeg.
In the command prompt, type `ffprobe /path/to/your/video/video.mp4`. The correct path to the video file should be specified.

Here is what the ffprobe output for an incorrect file should look like:

```
ffprobe version N-61916-g46f72ea Copyright (c) 2007-2014 the FFmpeg developers

Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'video.mp4':
  Duration: 00:05:00.18, start: 0.012000, bitrate: 769 kb/s
    Chapter #0.0: start 0.000000, end 300.000000
    Metadata:
      title           : Chapter 1
    Stream #0:0(eng): Video: h264 (High) (avc1 / 0x31637661), yuv420p, 640x360 [SAR 1331:1000 DAR 2662:1125], 636 kb/s, 23.98 fps, 23.98 tbr, 24k tbn, 47.95 tbc
(default)
    Metadata:
      handler_name    : VideoHandler
    Stream #0:1(rus): Audio: aac (mp4a / 0x6134706D), 48000 Hz, stereo, fltp, 128 kb/s (default)
    Metadata:
      handler_name    : SoundHandler
    Stream #0:2(eng): Subtitle: mov_text (text / 0x74786574)
    Metadata:
      handler_name    : SubtitleHandler
```

Here we can see several tracks, where `Video: h264` means using proper codec h264 and `Audio: aac` — proper codec aac.

Here is how the ffprobe output for an incorrect file should like:

```
Input #0, asf, from 'video.wmv':
  Duration: 00:05:00.22, start: 0.000000, bitrate: 388 kb/s
```

```
    Chapter #0.0: start 0.000000, end 300.217000
    Metadata:
      title           : Chapter 1
    Stream #0:0: Video: msmpeg4v3 (MP43 / 0x3334504D), yuv420p, 640x360, 23.98 tbr, 1k tbn, 1k tbc
    Stream #0:1: Audio: wmav2 (a[1][0][0] / 0x0161), 48000 Hz, 2 channels, fltp, 128 kb/s
```

Here we can see that the Windows Media format (wmv) is used with the corresponding codecs. Flussonic Media Server will not play anything back.

What should be done if the format is incorrect? To convert wmv/msmpeg/wma into mp4/h264/aac, ffmpeg may be used:

```
ffprobe /path/to/your/original/video/video.wmv /path/to/your/modified/video/video.mp4
```

The monitor will display something like:

```
Stream mapping:
  Stream #0:0 -> #0:0 (msmpeg4 -> libx264)
  Stream #0:1 -> #0:1 (wmav2 -> libvo_aacenc)
Press [q] to stop, [?] for help
frame=  937 fps=180 q=-1.0 Lsize=    2320kB time=00:00:39.04 bitrate= 486.7kbits/s dup=1 drop=0
```

This process is called transcoding, and it may be very time- and resource-consuming. The more powerful your hardware is, the faster transcoding is performed.

For the same reason, Flussonic Media Server does not transcode files automatically. It is assumed that users will do it manually on dedicated hardware. By the way, you cannot transcode live broadcasts in advance, therefore Flussonic offers a built-in transcoder for streams.

As a result of all the above we will receive a file that is completely suitable for playing back in Flussonic Media Server.

Preparing file: Picture quality

You might wish to lower the quality, or make a multi-bitrate file (which will ensure comfortable viewing by users connected to the Internet at different rates).

Learn more in file transcoding.

Configuring Flussonic Media Server

In order to make Flussonic Media Server start servicing files, add special configuration to the configuration file (`/etc/flussonic/flussonic.conf`):

```
vod vod {
  storage /movies;
  download;
}
```

This setting is called 'VOD location' (this word is used in the documentation and by our customer support team).

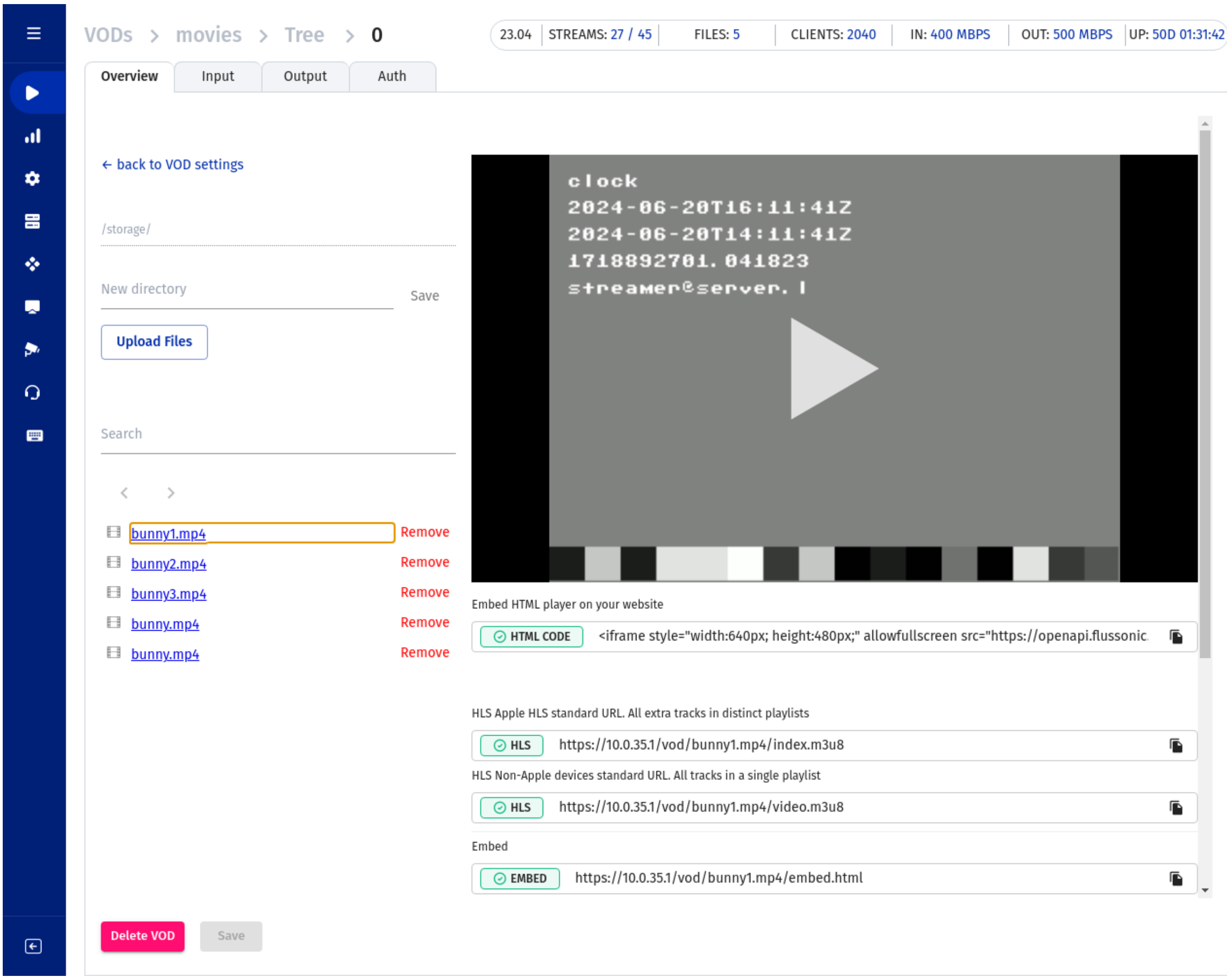
`/movies` is the folder on the server disk where video files are stored.

Technically this folder can be located in an NFS shared disk, but it's not a very good decision, since NFS is slow and not always good to use. One should better use a local hard disk, or an SSD.

After you add this setting, apply the changes in configuration by running the command `service flussonic reload`

Uploading file to server

To upload a file to the server, you can use web interface. In the main menu, select **Media** -> **VODs** -> location name -> **browse**. On the page that opens, you can add files to the VOD location, create nested directories to organize files, play files and get links for playing VOD files.



Note that using the web interface is not the only way to upload a file. The file can be uploaded using SSH or FTP, or any other way of transferring files over the network. The main thing is that the file is in the directory that is specified in the configuration file. Flussonic Media Server is executed as root, meaning it has access to any files, therefore, no special access rights to this uploaded file are required.

Viewing the file

In the web interface (in Media > VOD) click on the desired file to start playing it.

The URL of the video is indicated just below it, and may be used for watching outside the web interface.
It should look something like: `http://FLUSSONIC-IP/vod/elementary/s01e02.mp4/index.m3u8`
The URL ends with `.m3u8`, which means that the HLS protocol will be used for playback.

Such an URL may be watched in any player that supports HLS well. For example, you may download and run video player VLC, select **Media > Open Network Stream** or **Media > Open URL**, or press the key combination Ctrl+N and paste the URL into the input box.

Additional actions

Read the documentation about how VOD works in Flussonic
It contains answers to a few questions that are not covered in this article.

3.6.3 Preparing multibitrate files

Adaptive bitrate streaming ensures a good viewing experience for users with different connection capacities. To set up adaptive streaming, you need to create a multi-bitrate MP4 file and request a manifest file for it. Flussonic will do the rest.

The following contains detailed instructions on adaptive streaming setup and multi-bitrate file creation.

Installing tools

It is necessary to install ffmpeg and codecs. Note that the installation process differs depending on your OS.

INSTALLATION INSTRUCTIONS FOR WINDOWS

1. Download ffmpeg from https://ffmpeg.org/download.html
2. Follow the installation guide.
3. Download and install K-Lite Mega Codec Pack: http://www.codecguide.com/download_k-lite_codec_pack_mega.htm
4. Once the installer launches, select the fullest complete installation option ("Lots of stuff").

In case you have Windows 8.1, it is necessary to perform the following:

1. Press the Windows + Pause key combination.
2. Click on the Advanced system settings.
3. Click on the Environment Variables button.
4. Proceed to the System Variables.
5. Find the Path line.
6. Insert `C:\ffmpeg;C:\ffmpeg\bin;` to the beginning of the value.

INSTALLATION INSTRUCTIONS FOR UBUNTU LINUX

We recommend to use pre-built ffmpeg from this site: http://johnvansickle.com/ffmpeg Or any other pre-built binary from the official web site: https://www.ffmpeg.org/download.html

We don't recommend using ffmpeg from your Linux distro. It could be too old for transcoding h264, too old to work with our guides (or any other guides that rely on modern ffmpegs), or some other issues may occur.

Once the codecs installation is complete, your computer is ready to encode video.

Constructing an ffmpeg command to get multi-bitrate video

In this article, we will explore how to create a multi-bitrate stream using FFmpeg. As an example, we will take the video file `hall.mp4` and transcode it into multiple versions with different bitrates and resolutions, which is particularly useful for adaptive streaming.

ANALYZING THE SOURCE FILE

Before starting the encoding process, it is recommended to inspect the contents of the source video file:

```
ffmpeg -i hall.mp4
```

PREPARING FOR ENCODING

We will encode the video into five different resolutions with fixed bitrates. Two-pass encoding will be used to improve the final video quality.

FIRST ENCODING PASS

In the first pass, FFmpeg analyzes the video and collects statistics for optimal bitrate distribution:

```
ffmpeg -y -i hall.mp4 \
    -pass 1 \
    -map 0:0 -map 0:0 -map 0:0 -map 0:0 -map 0:0 -map 0:1 \
    -c:v libx264 -sc_threshold 0 -x264-params "nal-hrd=cbr" -g 60 -b_strategy 0 -forced-idr 1 \
    -c:a libfdk_aac -b:a 160k -ac 2 \
    -b:v:0 200k -maxrate:v:0 200k -minrate:v:0 200k -bufsize:v:0 200k -filter:v:0 scale=-2:240 \
```

```
    -b:v:1 500k -maxrate:v:1 500k -minrate:v:1 500k -bufsize:v:1 500k -filter:v:1 scale=-2:360 \
    -b:v:2 1000k -maxrate:v:2 1000k -minrate:v:2 1000k -bufsize:v:2 1000k -filter:v:2 scale=-2:480 \
    -b:v:3 3000k -maxrate:v:3 3000k -minrate:v:3 3000k -bufsize:v:3 3000k -filter:v:3 scale=-2:720 \
    -b:v:4 4000k -maxrate:v:4 4000k -minrate:v:4 4000k -bufsize:v:4 4000k -filter:v:4 scale=-2:1080 \
    -f mp4 /dev/null
```

- `-y` — automatically confirms file overwriting.
- `-pass 1` — first encoding pass (collecting statistics).
- `-map 0:0 -map 0:0 -map 0:0 -map 0:0 -map 0:0 -map 0:1` — selects input streams: 5 video streams and 1 audio stream.
- `-c:v libx264` — encodes video using the H.264 codec.
- `-g 60` — sets the GOP (Group of Pictures) size to 60 frames.
- `-b:v:N Xk` — specifies the bitrate for each video stream (ranging from 200k to 4000k).
- `-filter:v:N scale=-2:Y` — resizes each video stream (240p, 360p, 480p, 720p, 1080p).
- `-c:a libfdk_aac -b:a 160k -ac 2` — кencodes the audio stream in AAC with a 160k bitrate and stereo channels.
- `-f mp4 /dev/null` — in the first pass, no output file is created; statistics are stored in temporary files.

SECOND ENCODING PASS

```
ffmpeg -y -i hall.mp4 \
    -pass 2 \
    -map 0:0 -map 0:0 -map 0:0 -map 0:0 -map 0:0 -map 0:1 \
    -c:v libx264 -sc_threshold 0 -x264-params "nal-hrd=cbr" -g 60 -b_strategy 0 -forced-idr 1 \
    -c:a libfdk_aac -b:a 160k -ac 2 \
    -b:v:0 200k -maxrate:v:0 200k -minrate:v:0 200k -bufsize:v:0 200k -filter:v:0 scale=-2:240 \
    -b:v:1 500k -maxrate:v:1 500k -minrate:v:1 500k -bufsize:v:1 500k -filter:v:1 scale=-2:360 \
    -b:v:2 1000k -maxrate:v:2 1000k -minrate:v:2 1000k -bufsize:v:2 1000k -filter:v:2 scale=-2:480 \
    -b:v:3 3000k -maxrate:v:3 3000k -minrate:v:3 3000k -bufsize:v:3 3000k -filter:v:3 scale=-2:720 \
    -b:v:4 4000k -maxrate:v:4 4000k -minrate:v:4 4000k -bufsize:v:4 4000k -filter:v:4 scale=-2:1080 \
    -f mp4 hall_mbr.mp4
```

- `-pass 2` — second encoding pass.
- The output file `hall_mbr.mp4` is created with multiple video streams of different quality.

ENCODING WITH NVIDIA (GPU)

Example of using the NVIDIA hardware encoder (h264_nvenc):

```
ffmpeg -analyzeduration 11M -hwaccel cuvid -c:v h264_cuvid -vsync 2 -avoid_negative_ts disabled -i hall.mp4 -err_detect ignore_err -loglevel
repeat+level+verbose -spatial-aq 1 -aq-strength 8 -y  \
    -map 0:v:0 -map 0:v:0 -map 0:v:0 -map 0:v:0 -map 0:v:0 -map 0:a:0 \
    -profile:v high -preset:v slow -threads 0 \
    -c:v h264_nvenc -cbr 1 -rc cbr_hq -2pass 1 -bf 2 -b_ref_mode 2 -g 50 -b_adapt 0 -no-scenecut 1 -forced-idr 1 -strict_gop 1 \
    -c:a libfdk_aac -b:a 128k \
    -b:v:0 650k -maxrate:v:0 700k -minrate:v:0 650k -bufsize:v:0 700k -filter:v:0 "scale_cuda=640:360" -trellis 1 \
    -b:v:1 1000k -maxrate:v:1 1100k -minrate:v:1 1000k -bufsize:v:1 1000k -filter:v:1 "scale_cuda=768:432" -trellis 1 \
    -b:v:2 2250k -maxrate:v:2 2350k -minrate:v:2 2250k -bufsize:v:2 2250k -filter:v:2 "scale_cuda=960:540" -trellis 1 \
    -b:v:3 3000k -maxrate:v:3 3100k -minrate:v:3 3000k -bufsize:v:3 3M -filter:v:3 "scale_cuda=-1:720" -trellis 1 \
    -b:v:4 5000k -maxrate:v:4 5100k -minrate:v:4 5000k -bufsize:v:4 5M -filter:v:4 "scale_cuda=-1:1080" -trellis 1 \
```

```
    -f mp4 "hall_nvenc_mbr.mp4" \
    -max_muxing_queue_size 1024
```

- `-hwaccel cuvid -c:v h264_cuvid` – using the NVIDIA hardware decoder to decode the input video.
- `-c:v h264_nvenc` – using the NVIDIA NVENC encoder to encode the video.
- `-profile:v high` – setting the High profile for improved encoding quality.
- `-preset:v slow` – slow – balancing between encoding speed and output file quality.
- `-cbr 1 -rc cbr_hq` – using Constant Bitrate (CBR) mode with high quality.
- `-2pass 1` – two-pass encoding for better compression quality.
- `-bf 2 -b_ref_mode 2` – configuring bi-directional frames for more efficient encoding.
- `-g 50` – setting the Group of Pictures (GOP) size to 50 frames.
- `-b_adapt 0 -no-scenecut 1 -forced-idr 1 -strict_gop 1` – controlling GOP structure and scene cut handling.
- `-b:v:N Xk -maxrate:v:N Xk -minrate:v:N Xk -bufsize:v:N Xk` – setting bitrate, buffering, and limits for each video quality.
- `-filter:v:N "scale_cuda=W:H"` – scaling with CUDA hardware acceleration.
- `-c:a libfdk_aac -b:a 128k` – encoding the audio stream using AAC (libfdk_aac) with a bitrate of 128 kbps.
- `-max_muxing_queue_size 1024` – increasing the muxing queue size to prevent errors when writing to MP4.

Encoding a video segment

Sometimes you need to encode only a specific segment of your video stream. To do this, use the following parameters: `-ss 00:00:00 -t 00:05:00`. The value of the `ss` parameter specifies the start of the segment in seconds. The value of the `t` represents the segment's duration.

These parameters can be used with other commands. For instance:

```
ffmpeg -analyzeduration 11M -hwaccel cuvid -c:v h264_cuvid -vsync 2 -avoid_negative_ts disabled -i hall.mp4 -ss 00:00:00 -t 00:05:00 -err_detect ignore_err -
loglevel repeat+level+verbose -spatial-aq 1 -aq-strength 8 -y  \
    -map 0:v:0 -map 0:v:0 -map 0:v:0 -map 0:v:0 -map 0:v:0 -map 0:a:0 \
    -profile:v high -preset:v slow -threads 0 \
    -c:v h264_nvenc -cbr 1 -rc cbr_hq -2pass 1 -bf 2 -b_ref_mode 2 -g 50 -b_adapt 0 -no-scenecut 1 -forced-idr 1 -strict_gop 1 \
    -c:a libfdk_aac -b:a 128k \
    -b:v:0 650k -maxrate:v:0 700k -minrate:v:0 650k -bufsize:v:0 700k -filter:v:0 "scale_cuda=640:360" -trellis 1 \
    -b:v:1 1000k -maxrate:v:1 1100k -minrate:v:1 1000k -bufsize:v:1 1000k -filter:v:1 "scale_cuda=768:432" -trellis 1 \
    -b:v:2 2250k -maxrate:v:2 2350k -minrate:v:2 2250k -bufsize:v:2 2250k -filter:v:2 "scale_cuda=960:540" -trellis 1 \
    -b:v:3 3000k -maxrate:v:3 3100k -minrate:v:3 3000k -bufsize:v:3 3M -filter:v:3 "scale_cuda=-1:720" -trellis 1 \
    -b:v:4 5000k -maxrate:v:4 5100k -minrate:v:4 5000k -bufsize:v:4 5M -filter:v:4 "scale_cuda=-1:1080" -trellis 1 \
    -f mp4 "hall_nvenc_mbr.mp4" \
    -max_muxing_queue_size 1024
```

This is the aforementioned encoding command, but applied only to the first 5 seconds of the video clip.

Converting files for web streaming

Flussonic supports the following Containers and codecs. If your video file is encoded with a different codec, it will not play via Flussonic. For example, the file might have an old codec like Xvid or MPEG4-Video that is no longer supported by new versions of browsers. In such cases, file transcoding is required.

To convert any file to H.264, in the console change the directory to where you've put this file, and run the command:

```
ffmpeg -i input_file.avi -c:v libx264 -g 100 -c:a aac -f mp4 output_file.mp4
```

Similarly, to convert any file to H.265/HEVC:

```
ffmpeg -i input_file.avi -c:v libx265 -g 100 -c:a aac -f mp4 output_file.mp4
```

3.6.4 Multibitrate VOD streaming using SMIL

If you have several files with the same content and different bitrate, you can use SMIL files for adaptive bitrate VOD streaming.

SMIL file is a file in XML format that allows to make playlists for different combinations of files with different bitrates. It works similarly to `auto-mbr` functionality described here, but gives more flexibility: you can use not all files from the directory, but only the specified files. Also, there are no rules for naming the files, they just should be placed into the directory where the SMIL file is located or its subdirectories.

You can use SMIL-files for a VOD location on a local computer or in Amazon S3 storage. The example below is related to a local VOD location. If you are using Amazon S3 storage, follow the similar steps, but specify the URL of the storage in the VOD location settings as described here.

Configuring VOD location

Let's assume you have already created a VOD location.

Add the option `auto_mbr` to the VOD location that you want to use to store files for a multi-bitrate playlist.

- Via configuration file:

```
vod vod1 {
  storage /storage;
  auto_mbr;
}
```

- Via the web UI:

Go to **Files (VOD)** > open a location > go to the **Output** tab > select **Enable MBR from multiple files**.

Preparing files

Prepare the files with the same content and different bitrate, for example: `bunny_450`, `bunny_750`, `bunny_1100`. You can give the files any names.

Place the files into the same directory in the VOD location. Some files may be placed into subdirectories of this directory (e.g., you may place `bunny_110` into the `folder` subdirectory).

Creating SMIL file

Using a text editor, create a `*.smil` file (for example, `my.smil`) in the same directory, where the files are placed. The SMIL file should have the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<smil title="">
    <body>
        <switch>
            <video height="240" src="bunny_450.mp4" systemLanguage="eng" width="424">
                <param name="videoBitrate" value="450000" valuetype="data"></param>
                <param name="audioBitrate" value="44100" valuetype="data"></param>
            </video>
            <video height="360" src="bunny_750.mp4" systemLanguage="eng" width="640">
                <param name="videoBitrate" value="750000" valuetype="data"></param>
                <param name="audioBitrate" value="44100" valuetype="data"></param>
            </video>
            <video height="720" src="folder/bunny_1100.mp4" systemLanguage="eng" width="1272">
                <param name="videoBitrate" value="1100000" valuetype="data"></param>
                <param name="audioBitrate" value="44100" valuetype="data"></param>
            </video>
        </switch>
    </body>
</smil>
```

The only required and meaningful parameter for each file is `src` — the relative path to the file in the directory where the SMIL file is located. Flussonic ignores all other parameters and determines the size, language, and bitrate of the video automatically.

You can use any number of SMIL files in the directory to create playlists for different combinations of files.

Playback

You can request the playlist as described below.

- HLS playlist:

```
http://FLUSSONIC-IP/vod1/my.smil/index.m3u8
```

- DASH playlist:

```
http://FLUSSONIC-IP/vod1/my.smil/index.mpd
```

Flussonic creates an HLS or DASH playlist from multiple files listed in the SMIL file. The player works with this playlist as if it was one multi-bitrate file.

3.6.5 Streaming files from cloud

When deploying a Video on Demand (VoD) service with the file library size and the number of viewers unknown in advance it is advisable that you store your files in a cloud like Amazon S3/AWS, OpenStack Swift, Ceph, etc. Cold storage in the cloud can be cheaper than uniform disk storage, and cloud storage scales better as the service grows.

With *Flussonic Media Server*, you can broadcast video files kept on a cloud storage, as well as on HTTP servers.

Setting procedure in the UI

To setup safe and solid streaming from a cloud, you should:

1. Install *Flussonic Media Server* on a server with enough disk space for caching hot content. You can start with 256GB SSD, for example.
2. Make sure you have all the credentials required for accessing the storage. For example, you need to know `ACCESS_KEY` and `SECRET_KEY` for Amazon S3.
3. Create a VOD location in *Flussonic Media Server* at **Media -> VODs** with the storage address specified, for example:
   - For public Amazon S3 bucket: `http://s3.amazonaws.com/publicbucket`
   - For private Amazon S3 bucket: use your `ACCESS_KEY` and `SECRET_KEY` obtained from your Amazon AWS profile in the private storage URL like this: `http://ACCESS_KEY:SECRET_KEY@s3.amazonaws.com/privatebucket`
   - For Swift storage: `swift://user=USER&password=PASSWORD@swift-proxy/bucket`
   - For generic HTTP server: `http://example.com/prefix?key=12345`.

   You can use any parameters in the query string. When accessing a file, for example, `http://FLUSSONIC-IP/vod/bunny.mp4`, the query to the storage will be `http://storage/prefix/bunny.mp4?key=12345`.

4. Configure a local file cache on the **Input** tab.

> **Note**
>
> Cache is recommended to smooth out the requests to remote storage. Without a cache, reading each frame will generate several network requests, while with cache enabled requests will be aggregated.

5. Then click **Browse** next to the path on the **Overview** tab and test the files playback.

VOD settings in the config file

The same settings may ba done via `/etc/flussonic/flussonic.conf` config file or via **Config editor** in the UI. An example of VOD configuration is given below.

Note that cache is configured to avoid frequent requests to the cloud.

```
vod public {
  storage http://s3.amazonaws.com/publicbucket;
  cache /cache 100G;
}
vod private {
  storage s3://ACCESS_KEY:SECRET_KEY@s3.amazonaws.com/privatebucket;
  cache /cache 100G;
}
```

Related objectives

- How to View a File: streaming from disk.

3.6.6 A multibitrate playlist made from files

Creating multibitrate content from multiple files

Suppose you have copies of a movie in several files with different qualities. Furthermore, you do not want to create one multi-bitrate file. You need to play these files by using a single HLS or DASH playlist so that the client player can choose the bitrate the same way as with one multi-bitrate file.

Flussonic Media Server can deliver several files with different bitrates as a single resource with multi-bitrate content. The HLS or DASH playlists contain information about these files as if it was one file in various qualities.

You should preprocess the files first and then enable the automatic generation of a multi-bitrate resource for a VOD location.

You can use VOD location on a local computer or in Amazon S3 storage. The example below is related to a local VOD location. If you are using Amazon S3 storage, follow the similar steps, but specify the URL of the storage in VOD location settings as described here.

PREPARE FILES

Place the files in the same directory. Give them names that start with the name of the directory in which they are located. That is, file names must match the `DIRNAME*.mp4` mask, where `*` stands for any allowed characters. For example:

Directory name: **DIR_NAME**, file names: **DIR_NAME-1.mp4**, **DIR_NAMEabc.mp4**, and so on.

See **Step 2** below.

SET UP AUTOMATIC CREATION OF A MULTIBITRATE RESOURCE

Let's assume you have already created a VOD location for accessing the files.

Step 1. Add the option `auto_mbr` to the VOD location that you want to use to store files for a multi-bitrate playlist.

- Via configuration file:

```
vod vod1 {
  storage /storage;
  auto_mbr;
}
```

- Via the web UI:

Go to **Files (VOD)** > open a location > go to the **Output** tab > select **Enable MBR from multiple files**.

Step 2. Place files in the directory, for example:

`/storage/movies/bunny/bunny.480x360.mp4`

`/storage/movies/bunny/bunny.720x480.mp4`

`/storage/movies/bunny/bunny.1080x720.mp4`

Flussonic determines the size of the video, so it is unnecessary to specify the size in the file name. You can use an arbitrary set of valid characters after the word `bunny` in file names.

Step 3. Now you can request an:

- HLS playlist:

```
http://FLUSSONIC-IP/vod1/bunny/index.m3u8
```

- DASH playlist:

```
http://FLUSSONIC-IP/vod1/bunny/index.mpd
```

You can see the playlist is requested on a directory, not a single file.

When a playlist is requested on one of the directories: `/vod/bunny/index.m3u8` or `/vod/bunny/index.mpd`, *Flussonic* creates an HLS or DASH playlist from multiple files matching the mask `/vod/bunny/bunny*.mp4`. The player works with this playlist as if it was one multi-bitrate file.

Note

Clients can read the contents of only those directories for which the `auto_mbr` option is specified in the settings. Otherwise, *Flussonic* will return a `404 error`.

3.6.7 Cache

To speed up the broadcasting of VOD, you can use the SSD cache.

To configure caching, for original files from the cloud or HTTP server the `cache` option is used.

For your files on SSD drives, you can use intermediate SSD caching of video file segments. The option for this operation is called `segment_cache`.

File caching on SSD

You can ask Flussonic Media Server to save not chunks, but file content on disk, when the source is a cloud or a remote HTTP server (such as another Flussonic).

This mechanism can allow you to build a distributed CDN from several Flussonics because now even downloading will lead to caching a whole file.

Flussonic Media Server will not download the same content twice, so simultaneous access to a file is collapsed into a single upstream request.

> **Warning**
>
> For file cache, do **not** use SSD partitions that were mounted with the option `noatime`.

Here is the configuration for file cache:

```
vod vod_remote {
  storage s3://minioadmin:minioadmin@minio:9001/test;
  cache /storage/cache 400G;
  download;
}
```

Such configuration will download files on `/mount/ssd` on request: only requested data will be available locally.

CACHING BASED ON THE NUMBER OF REQUESTS

You can define a condition for placing files in cache - this condition is how often a file was requested by clients.

The option **misses=3** tells Flussonic that if this file was requested more than 3 times, it must be cached:

```
vod vod_remote {
  storage s3://minioadmin:minioadmin@minio:9001/test;
  cache /storage/cache 400G misses=3;
  download;
}
```

CHOOSING CACHE OPTIONS IN THE UI

To set cache options for file in the Flussonic UI:

1. Proceed to the **VODs** and click on the file that you want to cache.
2. Open the **Input** tab and edit the **Cache** section.

Segment cache for SSD

Today, one of the most popular ways to speed up serving content from a disk is using SSD storage.

Since solid state drives cost significantly higher than traditional HDDs, quite often it makes sense to use the setup that involves intermediate SSD caching.

Flussonic Media Server can automatically cache the requested chunks for HLS on a disk, which allows to speed up delivery considerably. Specify the following configuration:

```
vod vod1 {
    storage /mount/hdd1;
```

```
    storage /mount/hdd2;
    storage /mount/hdd3;
    segment_cache /mount/ssd1 20G 48h misses=2;
}
```

With this configuration, Flussonic Media Server maintains the cache size limit of 20GB, deletes files older than 2 days, and caches only the files requested more than twice. The option `segment_cache` is specified once only.

> **Warning**
>
> We do not recommend using traditional hard drives (HDDs) for `segment_cache`, use SSD instead.

3.7 Push

3.7.1 Publishing to social media

Flussonic Media Server allows you to publish any stream to an external server using RTMP.

Social media use RTMP to organize live broadcasts, which means that you can use Flussonic Media Server to send your streams to social media (it can be several at once).

Scenarios for use:

- Receiving video from a mobile reporter and sending directly to several social media.
- Broadcast video from CCTV cameras.
- Broadcast their own programs in social media. Including on schedule.

> **Warning**
>
> Stream keys can expire. Check the terms of the service before publishing the stream.

Content:

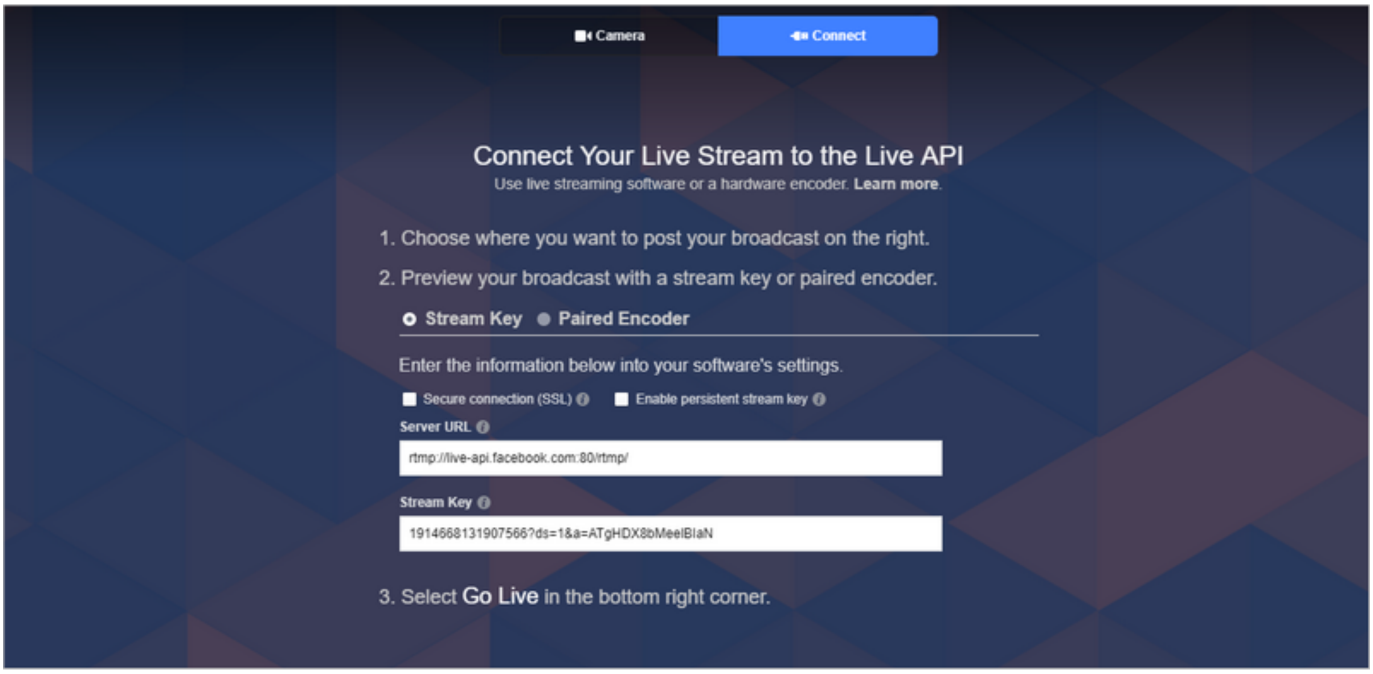
- Publish to Youtube
- Publish to Facebook
- Publish to OK

Publish to YouTube

See Restreaming to YouTube in high quality.

Publish to Facebook

1) Go to **Facebook** > **Live Videos** > **Go Live** > **Connect**.



2) Copy the server URL and the stream key.

3) In Flussonic Media Server administrative interface, go to **Media** menu and select the stream you want to distribute.

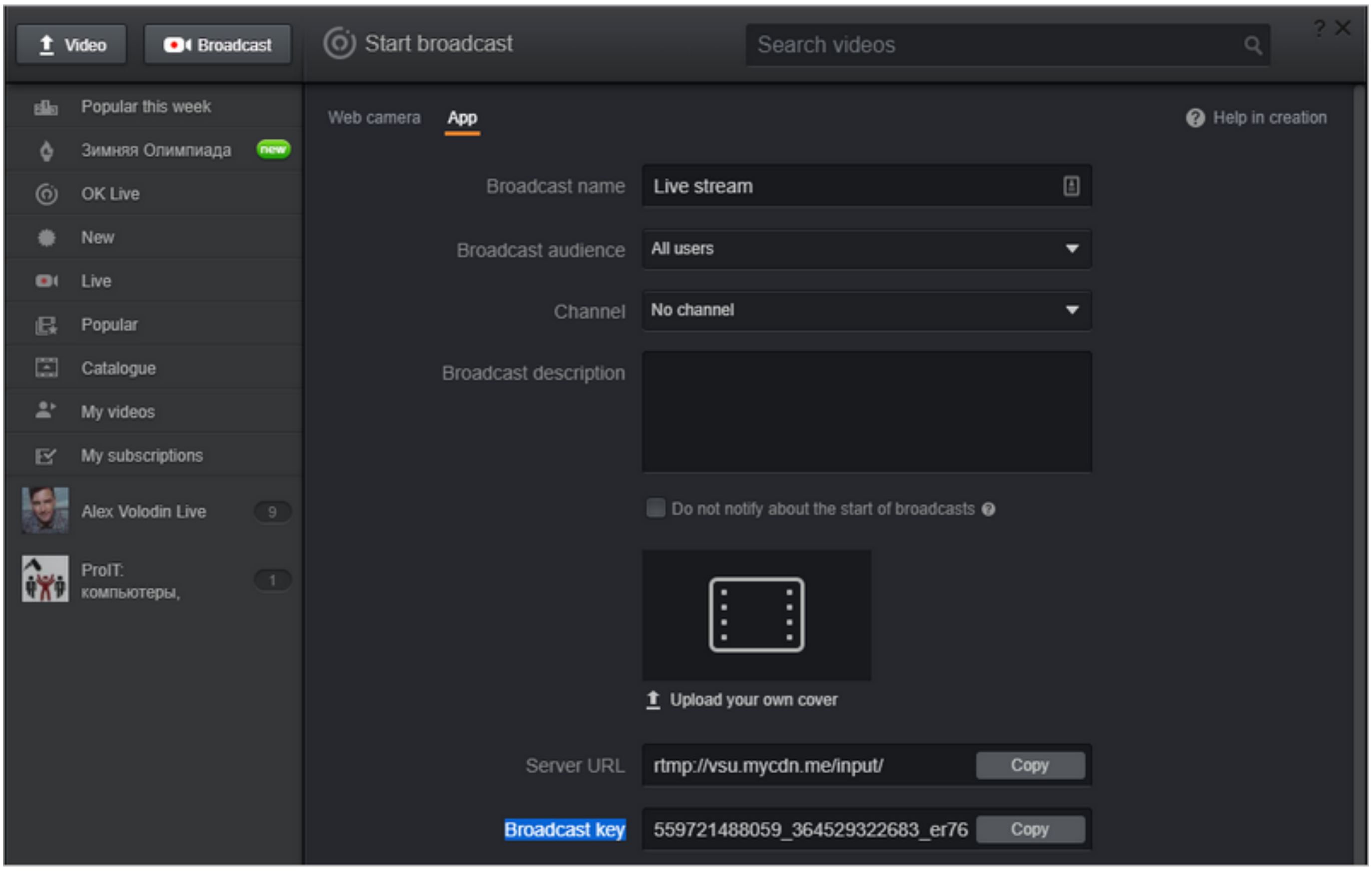
4) In the tab **Output**, find the **Push live video to certain URLs**.

5) Paste the URL of the server and the stream key as a link. For example, `rtmp://live-api.facebook.com:80/rtmp/1917254653482108?ds=1&a=ATj3ccSijhehV15i`. Press **Save**.

6) Return to **Facebook** > **Live Videos** > **Go Live** > **Connect** and start the live broadcast.

Publish to OK

1) Go to **OK.ru** > **Broadcast** > **App**.



2) Copy the server URL and the broadcast key.

3) In Flussonic Media Server administrative interface, go to **Media** menu and select the stream you want to distribute.

4) In the tab **Output**, find the **Push live video to certain URLs**.

5) Paste the URL of the server and the stream key as a link. For example, `rtmp://vsu.mycdn.me/input/5654546560699_3670934550827_rldbynpfqu`. Press **Save**.

6) Return to **OK.ru** > **Broadcast** > **App** and start the live broadcast.

3.7.2 Restreaming to YouTube in high quality

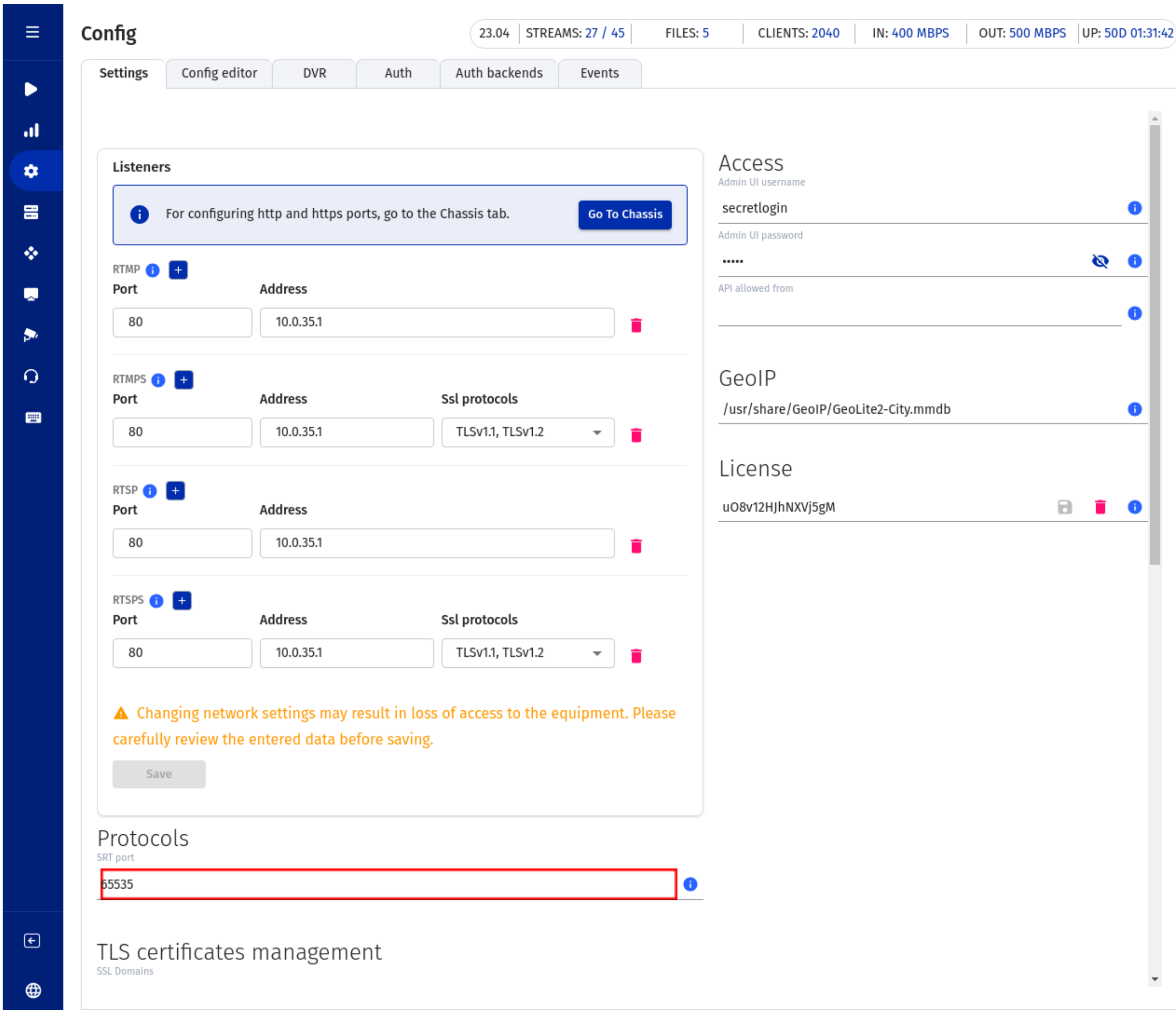
Use Enhanced RTMP to push high quality HEVC or AV1 encoded video to YouTube. Learn more about Enhanced RTMP.

Flussonic Media Server can push any stream via Enhanced RTMP, for example, a WebRTC-published, an IP camera, a server playlist, etc. On this page, you will find an example of restreaming an Enhanced RTMP stream.

Step 1. Getting Flussonic ready for publishing via Enhanced RTMP

Configure Flussonic for publishing:

1. On the **Config** -> **Settings** tab, set port 1935 for RTMP publishing.

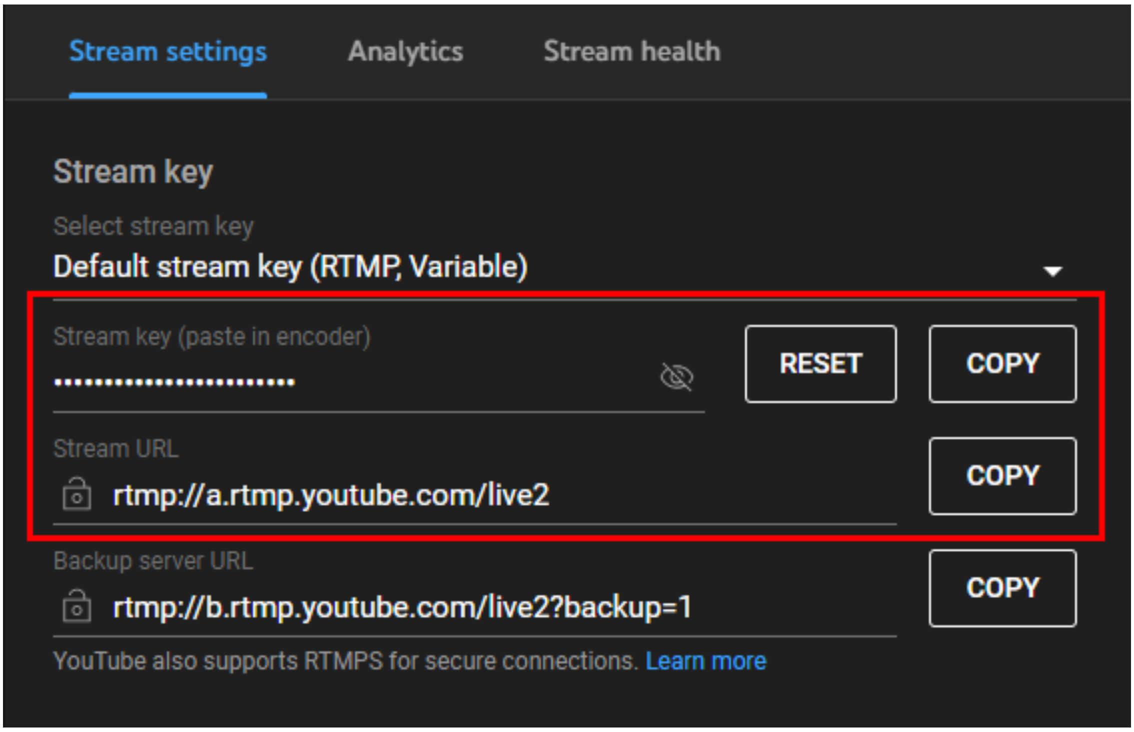


2. Create new stream by clicking **+** on the **Media** -> **Streams** page. In the stream creation form, enter stream name, for example `published`, and set the **Publication** checkbox.

Step 2. Getting URL for sending video to Youtube

1. Go to YouTube Studio > **Content** > **Live** and click **GET STARTED**. The stream settings are displayed.

2. You will need the server URL and the stream key.



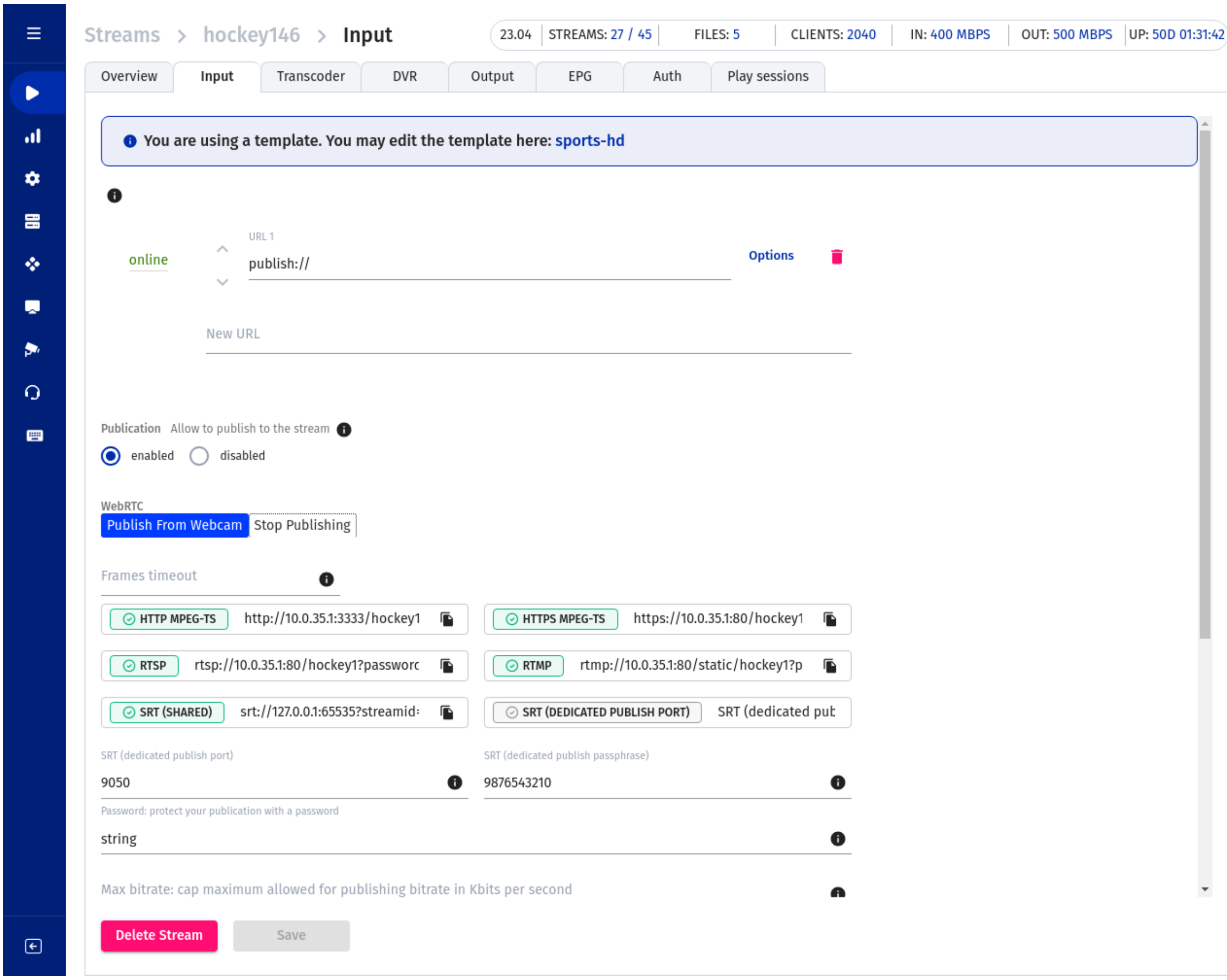
Step 3. Pushing from Flussonic via Enhanced RTMP

Configure sending video from Flussonic to YouTube:

1. Go to **Output** tab of the `published` stream you have created on step 1.
2. In the **Push video to certain URLs** section, paste the translation URL and key copied from YouTube Creator Studio. For example, `rtmp://a.rtmp.youtube.com/live2/7p9v-6gsh-18jm-223h`. Press **Save**.

Step 4. Restreaming

1. Go to the **Input** tab of the `published` stream. Copy the **RTMP** URL in the **Publish links: start publishing to get preview** group.



2. Publish to Flussonic from any app supporting Enhanced RTMP, for example with ffmpeg:

> **Warning**
>
> You need ffmpeg 6.1.1 or higher version for publishing via Enhanced RTMP.

```
ffmpeg -i /opt/flussonic/priv/bunny.mp4 -c:v libx265 -preset fast -c:a copy -f flv rtmp://localhost:1935/static/published
```

If everything is configured correctly, you will see the file broadcast on your YouTube channel.

What if the stream has no audio?

YouTube's pipeline requires audio in all videos. It may happen that your video has no audio, for example, if it is a stream from an IP camera or published via WebRTC from a browser. Flussonic Media Server can add an empty, silent audio track to the stream so that it plays normally on YouTube.

To add the silent track to the stream:

1. Go to **Input** tab in the stream profile and click **Options** next to the input URL.
2. Select **Add_AAC** in the **Output audio** list.

3.7.3 Pushing a stream to other servers

How to copy (push) a stream to other servers

You can tell Flussonic to copy a stream to other servers by using the `push` directive. For example, you can push a stream to a CDN or to social media.

To use this function, go to the **Output** tab in the stream settings and scroll down to the **Push live video to certain URLs** section. Here you can add URLs to which *Flussonic* will push the stream.

Flussonic supports push over the following protocols:

RTMP

This protocol is usually utilized for pushing streams to the social networks. Please refer here for more details and examples.

HTTP MPEG-TS

This protocol is suitable for sending streams to third-party video streaming services. The publish link must be provided by the service you are pushing to.

HLS

You can push streams to a cloud or CDN using this protocol; usually CDNs support it. Below is an example of configuration for sending a stream to the Amazon AWS cloud.

Example: pushing a stream to Amazon AWS

```
stream breakingnews {
  input publish://;
  segment_count 10;
  segment_duration 10;
  push hlss://[api-id].execute-api.[aws-region].amazonaws.com/[stage]/[folder-name];
}
```

This config will allow you to use Amazon AWS API Gateway as a service proxy to Amazon S3 as described in Amazon's documentation.

M4S

Use this protocol to push streams to another Flussonic server. This is a persistent protocol that does not create delays and is used to transfer data from Flussonic to Flussonic for further delivery via WebRTC/RTMP.

M4S URL can be composed like that: `m4s://FLUSSONIC-IP:PORT/STREAM_NAME` .

Pushing with a 302 redirect

When publishing via m4s:// Flussonic will understand HTTP 302 and will follow to the specified address. This means that you can specify not only the Flussonic server address, but also your own backend for choosing a publishing location. For example, `m4s://example.com/router;` .

How to manage pushed streams

If the stream is configured to be pushed to an URL that has become offline, then Flussonic by default endlessly retries to push the stream to this URL.

Flussonic can monitor streams that it sends to other servers and collect statistics on unsuccessful sending attempts. A visual display of the push statuses in the UI will help you take action — stop (pause) offline streams or limit attempts to send them.

Push statuses are shown as indicators on the main page in the **Streams** list and in stream settings on the **Output** tab (**Push live video to certain URLs**). The reason for stopping the pushing process can be found in the logs.

3.7.4 Pushing SRT stream

Flussonic supports pushing SRT streams. Pushing streams via the SRT protocol is widely used when delivering video over the Internet or satellite network, because SRT guarantees low latency while offering some content delivery guarantees. Read more about SRT at Using SRT protocol.

Let us have a look at the configuration options.

Push from Flussonic

You can configure pushing SRT streams from *Flussonic* in the same way as for any other protocol as described on the Pushing a Stream to Other Servers page. For SRT, you should set the URL according to one of the following formats:

- SRT parameters in the URL parameters:

  `srt://SRT-HOST:SRT_PORT streamid="#!::r=STREAM_NAME,m=publish"`

- SRT parameters in the URL query string:

  `srt://SRT-HOST:SRT_PORT?streamid=#!::r=STREAM_NAME,m=publish`

where:

- `SRT-HOST` is an IP address of the destination server.
- `SRT_PORT` is an SRT port.
- `streamid` is a string formatted as described here.
- `STREAM_NAME` is a name of a publishing location to push the SRT stream to.

Let's have a look at the example:

```
stream push_srt {
  input fake://fake;
  push srt://example.com:9998 streamid="#!::r=my-stream-id,m=publish";
}
```

In the example above we enabled stream transmission (`push`) to the `example.com` server over port `9998` .

Parameters for SRT push

You can also manage SRT push by passing certain parameters.

Example:

```
stream srt_push {
  input fake://fake;
  push srt://example.com:9998?streamid=#!::r=some_random_name&passphrase=1234567890;
}
```

3.7.5 Sending multicast

When working with IPTV, one often has to deal with videos transmitted as multicasts. In most cases, a multicast contains an MPEG-TS container (7 188-byte packets in each UDP packet). Less frequently, the RTP video in transmitted into the network that contains the same MPEG-TS. RTP is needed to make it possible to track the losses, since the RTP packet contains a 16-bit counter that is used to track sequence numbers.

Brief basics of multicast

A multicast is a set of UDP packets distributed from the same source to a group of subscribers. The address to which packets are sent is usually in the range between 224.0.0.0 and 239.255.255.255, however, 224.0.0.0/8 is not recommended due to the large number of special addresses.

In a properly configured network, multicast traffic is sent to the nearest router, and the router itself chooses the client to send the traffic to, based on the requirements of the clients. The requirements are transmitted via the IGMP protocol that is used for transmitting messages about the need to include some address into the distribution group, or exclude it from the group.

Therefore, in order to make *Flussonic* send multicast to client devices, it is necessary to make it send the packets to the proper interface (in a local operator network), and the router should be configured to work correctly with multicast.

Ingesting of multicast streams is described in Receiving multicast.

Prerequisites

Make sure *Flussonic* synchronizes time with an NTP server. This is important for multicast stability so that the timestamps in the input and output streams were synchronized.

Configuring Flussonic

You can configure multicast streaming in the configuration file or in the web interface.

To configure multicast streaming in the configuration file, set the `push` option in the stream settings and specify the multicast address:

```
stream origin {
  input fake://fake;
}
stream example {
 input hls://localhost:80/origin/index.m3u8;
 push udp://239.0.0.1:1234;
}
```

To configure multicast streaming using the web interface:

1. Create a new stream and specify the source URL on the **Input** tab.
2. Go to the **Output** tab in the stream settings and specify the multicast address (like `udp://239.0.0.1:1234`) in the **Push live video to certain URLs** section.

3. (Optional) Set the average bitrate, bitrate, PMT, PNR, standby, multicast loop parameters for an MPEG-TS stream by clicking the **Options** button. Specify values for the parameters and click **Save** to apply changes. You can do this in the template settings as well.

Selecting tracks

You can select what tracks to send:

```
stream origin {
  input fake://fake;
}
stream example {
 input hls://localhost:80/origin/index.m3u8;
 push udp://239.0.0.1:1234?tracks=v1a1;
}
```

Here, `v1` stands for the 1st video track and `a1` for the 1st audio track.

Maximum bitrate

Flussonic can push multicast with maximum bitrate value in PMT (Program Map Table) for every ES (Elementary Stream). To enable this option you have to add the `es_max_bitrate=default` string to the query string. The configuration may look as follows:

```
stream example {
  input file://vod/STREAM_NAME.ts;
  push udp://239.0.0.1:1234?cbr=6000&tracks=v1a1&es_max_bitrate=default;
  transcoder vb=5000k fps=25 preset=fast hw=cpu ab=192k;
}
```

Interface name

If you do not remember the IP address of the interface from where the multicast will be sent, you can specify its name:

```
push udp://eth0@239.0.0.1:1234
```

instead of

```
push udp://239.0.0.1:1234/10.0.0.5
```

Example:

```
stream example {
  input hls://provider.iptv/stream/index.m3u8;
  push udp://eth0@239.0.0.1:1234;
}
```

Here `eth0` is the name of the interface that looks into a local network.

Looping back a multicast stream to the Flussonic host

If you push a stream from *Flussonic* to UDP multicast, you can use the multicast socket option `multicast_loop` that enables ingesting the sent UDP data back to the *Flussonic* host:

```
stream example_push {
  input hls://provider.iptv/stream/index.m3u8;
  push udp://239.0.0.1:1234 multicast_loop;
}

stream example_ingest {
  input udp://239.0.0.1:1234;
}
```

This option allows you to ingest the sent stream on the sending host by *Flussonic* or other application.

Configuring the server

After you set up multicasting, chances are that nothing will work, since very often, due to server settings, multicast traffic is sent to the first interface, which usually looks into the Internet. You need to make *Flussonic* start sending traffic to an interface that looks into a local network.

```
route add -net 239.0.0.0/8 dev eth2
```

Here, `eth2` is the name of the interface connected to the local network. After you set up routing in this way, the multicast from *Flussonic* will be routed to the proper interface, and you can check it at the router, and at the client.

Specifying PIDs

When sending MPEG-TS to UDP multicast (`push udp://`), specify PIDs by using the mpegts_pids option.

Another way to specify PIDs is as follows:

```
stream example {
  input hls://provider.iptv/stream/index.m3u8;
  push udp://239.1.2.4:1235 bitrate=7000 pnr=2 vb=6000 pmt=2000 v1=2011 a1=2021;
}
```

Signalling AC-3 audio stream in MPEG-TS

MPEG-TS containing AC-3 elementary audio stream is regulated by the STD (System Target Decoder) model in System A (ATSC) or System B (DVB). Signaling formats in System A (ATSC) and System B (DVB) vary substantially. So uniquely identifying AC-3 streams is used not only to indicate unambiguously that an AC-3 stream is an AC-3 stream, but to which System (A or B) the stream belongs.

Flussonic can read different signaling formats of AC-3 elementary stream in PMT. *Flussonic* can either pass the original signaling format of AC-3 stream in MPEG-TS to UDP multicast or convert the format to comply with System A or System B with the help of the `mpegts_ac3` option. This option is specified in the stream/template settings, and takes the following values:

- `mpegts_ac3=keep` — keeps the original AC-3 audio stream signaling format passing it to the output
- `mpegts_ac3=system_a` — modifies the AC-3 signaling format to match System A
- `mpegts_ac3=system_b` — modifies the AC-3 signaling format to match System B

> **Note**
>
> If an input and output audio is in AC-3 format, you don't have to enable transcoding to use the `mpegts_ac3`.

Consider the following configuration:

```
stream example-stream {
  input udp://MULTICAST-IP-1:PORT-1 programs=2;
  push udp://MULTICAST-IP-2:PORT-2 bitrate=6800 mpegts_ac3=keep;
}
```

Here *Flussonic* ingests a UDP stream with AC-3 audio and pushes the UDP stream with the original audio signaling format to the output.

3.8 Playback

3.8.1 Video playback

Protocols for video playback

Flussonic Media Server can play video streams via various protocols. Click the stream name on the **Media** — **Streams** tab, then go to the **Output** tab to see all available playback URLs. You can copy each URL to the clipboard by clicking the "Copy" button on the right next to the corresponding URL.

Below you will find a bit more detailed descriptions of the URL addresses that you should use in players in order to play video via different protocols with links to sections that provide more information about configuring the playback via each certain protocol. You can also play a stream in the Preview Player directly in the *Flussonic UI*.

Additionally, you can manage video playback via the Streaming API.

EMBED.HTML

URL: `http://FLUSSONIC-IP/STREAMNAME/embed.html`

Flussonic Media Server has a special page embed.html which is intended for video insertion to a website or viewing of video via a browser. The page automatically detects a browser version to select a supported protocol. For the majority of devices for today — it's HLS. Read more in the Video insertion on the website (embed.html) article.

The complete **OpenAPI** specification: Streaming API.

HLS

URL for the player: `http://FLUSSONIC-IP/STREAMNAME/index.m3u8`

Read more in HLS playback. Use (embed.html)or any third-party player to insert HLS stream on your website. For example, hls.js or clappr.

The complete **OpenAPI** specification: Streaming API.

DASH

The stream is available at the URL: `http://FLUSSONIC-IP/STREAMNAME/index.mpd`

Read more in DASH playback.

The complete **OpenAPI** specification: Streaming API.

MSE-LD

URL for the player: `ws://FLUSSONIC-IP/STREAMNAME/mse_ld`

HTML5 (MSE-LD)

The stream played through HTML5 is available at the URL: `http://FLUSSONIC-IP/STREAMNAME/embed.html?realtime=true`

Read more in HTML5 (MSE) low latency playback.

MSS

The stream is available at the URL: `http://FLUSSONIC-IP/STREAMNAME.isml/manifest`

Read more in MSS Playback.

The complete **OpenAPI** specification: Streaming API.

HTTP MPEG-TS

The stream is available at the URL: `http://FLUSSONIC-IP/STREAMNAME/mpegts`

HTTP MPEG-TS relative timeshift

The URL for HTTP MPEG-TS playback with relative timeshift:

`http://FLUSSONIC-IP:PORT/STREAM_NAME/timeshift_rel-3600.ts`

In this example, the recorded stream will be played with one hour (3600 seconds) delay.

HTTP MPEG-TS absolute timeshift

The URL for HTTP MPEG-TS playback with absolute timeshift:

`http://FLUSSONIC-IP:PORT/STREAM_NAME/timeshift_abs-1643257722.ts`.

A fragment of an archive can be retrieved not on the full speed, but in the streaming mode, over a time equal to the length of the fragment.

RTMP

The stream is available at the URL:

- `rtmp://FLUSSONIC-IP/static/STREAMNAME`

RTSP

The stream is available at the URL:

- `rtsp://FLUSSONIC-IP/STREAMNAME`

If an RTSP stream has several audio and video tracks and you need to specify the tracks for playback, use the `filter.tracks` parameter.

See the following examples:

- `rtsp://FLUSSONIC-IP/STREAMNAME?filter.tracks=a2v1`
- `rtsp://FLUSSONIC-IP/vod/file?filter.tracks=a2v1` — VOD.
- `rtsp://FLUSSONIC-IP/STREAMNAME2 = rtsp://FLUSSONIC-IP/STREAMNAME1?filter.tracks=v1a1`

Selecting only one track:

- `rtsp://FLUSSONIC-IP/STREAMNAME?filter.tracks=a1` — select an audio track.
- `rtsp://FLUSSONIC-IP/STREAMNAME?filter.tracks=v1` — select a video track.

WEBRTC

WebRTC WHEP is available at the URL:

- `ws://FLUSSONIC-IP/STREAM_NAME/whep`

Read more about our WebRTC player and how to organize playback in WebRTC Playback.

The complete **OpenAPI** specification: Streaming API.

SHOUTCAST

The stream is available at the URL: `http://FLUSSONIC-IP/STREAMNAME/shoutcast`

Flussonic Media Server can deliver SHOUTcast, ICEcast radio stream.

The complete **OpenAPI** specification: Streaming API.

SRT

Flussonic supports playing SRT streams.

The general URL for playing an SRT stream looks like that:

`srt://FLUSSONIC-IP:SRT_PORT?streamid=#!::r=STREAM_NAME,m=request`

Learn more about SRT, `streamid` and other supported parameters at Using SRT protocol.

Possible URL options are listed at SRT Playback.

Filtering tracks for playback

If a stream has several audio, video, or subtitle tracks, you can specify certain tracks for *Flussonic* to deliver.

To do this this, add the `?filter.tracks=[aN|vN|tN|lN]+` parameter, which is followed by one or more track numbers without spaces to the end of the stream URL. Here are the possible values:

- `aN` is the Nth audio track
- `vN` is the Nth video track
- `tN` is the Nth WebVTT subtitles track
- `lN` is the Nth track with DVB subtitles or teletext.

By default, *Flussonic* uses the first video and audio tracks. If you specify more than two tracks or make a syntax error, the default tracks will be used (a1v1).

The following examples show how to use the parameter:

- `rtsp://FLUSSONIC-IP/STREAMNAME?filter.tracks=a2v1` : select the second audio track and the first video track for the RTSP stream.
- `http://FLUSSONIC-IP/STREAM_NAME/index.m3u8?filter.tracks=v2` : select the second video track for the HLS playlist.
- `http://FLUSSONIC-IP/STREAMNAME.isml/manifest?filter.tracks=v1t1t2t3` : select the first video track and three tracks with subtitles for the MSS manifest.

Restricting certain protocols

By default, playback via all protocols is allowed, but you can prohibit playback via the selected protocols (**Except**) or allow only certain protocols (**Only**) with the switch to the left of each of the URLs on the **Output** tab. This feature is useful not only for security, but also for reducing the load on the server: packing into all available protocols can consume a lot of resources.

These settings can be done in the config file as well. For example, you can allow all protocols except MPEG-TS and HLS for the stream `channel_01` (corresponds to **Except** switch position):

```
stream channel_01 {
  protocols -mpegts -hls;
}
```

To allow only DASH playback and API requests for the stream `channel_02` and disable other protocols (corresponds to **Only** switch position):

```
stream channel_02 {
  protocols dash api;
}
```

Getting data about played stream

You can make API requests to receive the information about the played stream. You can integrate the received data into any external system like site, monitoring, player, or mobile application.

The URL for getting technical information about the output media content:

`http://FLUSSONIC-IP/STREAM_NAME/media_info.json`

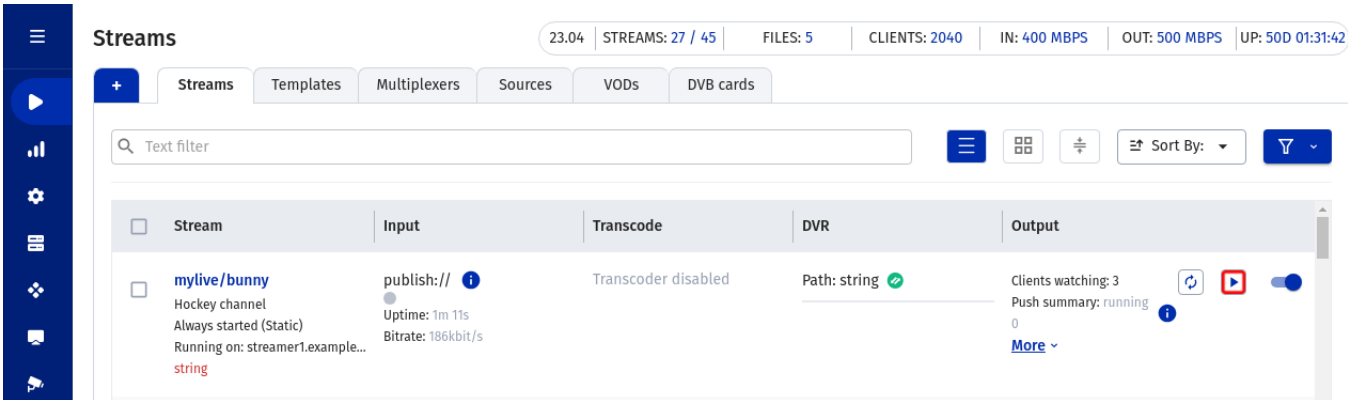
The URL for getting the information about DVR recording status of a stream:

`http://FLUSSONIC-IP/STREAM_NAME/recording_status.json`

Preview player in Flussonic UI

You can play a stream directly in *Flussonic UI* using the Preview Player. This player allows to play a stream via HLS, MSE-LD, or DASH protocol. Additionally, DVR playback is available for streams with DVR enabled. The Preview Player uses the special **embed.html** page (for details, see Adding Video to Websites) with corresponding parameters.

To open the Preview Player, go to **Media** > **Streams** and click the **Play** button next to the corresponding stream.



In the opened window choose one of the tabs and start playing. The following tabs are available:

- **HLS** — for HLS playback. The player will use `embed.html` page.
- **MSE** — for HTML (MSE-LD) low latency playback. The player will use `embed.html?realtime=true` page.
- **DASH** — for DASH playback. The player will use `embed.html?proto=dash` page.
- **DVR** — for DVR archive playback (if DVR is enabled). The player will use `embed.html?dvr=true` page. You can read more about the settings of DVR player in the Viewing the DVR recordings from the web interface chapter.

In the bottom of the Preview Player window you can see HTML code for inserting the corresponding player into a web page.

To close the Preview Player window, press **Esc**.

3.8.2 HLS Playback

Flussonic Media Server supports playing video via the HLS protocol. Many of Flussonic's HLS features use non-standard extensions of HLS – we support them for your convenience.

The supported codecs are: H264, H265, MPEG2 video, AAC, MP3, MPEG2 audio, and AC-3.

Flussonic Media Server supports access via HLS to live streams, VOD files and DVR (catchup and timeshift).

If an incoming stream has DVB subtitles or teletext, then Flussonic can pass them to an output HLS stream if you configure Flussonic for that. Subtitles are saved in DVR archive if a stream is recorded to the archive.

On this page:

Structure of HLS protocol

HLS protocol works by breaking the overall stream or file into a sequence of media segments of equal length. The segments are represented by `.ts` files sequentially downloaded by HTTP. HLS also creates an index file that contains references of media segment files, saved as `.m3u8`.

Depending on usage scenario, you can use different URLs for HLS playback, but in any case, the URL will result in one of the following two types of playlists (or manifests).

MEDIA PLAYLIST

Media Playlist is a playlist where all lines identify media segments. Each media segment is specified by a URI and has the `.ts` extension. The playlist continues to be updated as new media URIs are added. Such a playlist is used for playing simple streams or files with only one video track and one audio track.

Media Playlist example:

```
#EXTM3U
#EXT-X-TARGETDURATION:7
#EXT-X-VERSION:3
#EXT-X-MEDIA-SEQUENCE:38530
#EXT-X-PROGRAM-DATE-TIME:2022-03-22T08:06:37Z
#EXTINF:5.000,
2022/03/22/08/06/37-05000.ts
#EXTINF:5.000,
2022/03/22/08/06/42-05000.ts
#EXTINF:5.000,
2022/03/22/08/06/47-05000.ts
#EXTINF:5.000,
2022/03/22/08/06/52-05000.ts
```

MASTER PLAYLIST

Master Playlist is a playlist where all lines identify Media Playlists. Each Media Playlist is specified by a URI and has the `.m3u8` extension. Master playlist may be used for multibitrate playback and allows clients to switch between bitrates dynamically.

Master Playlist example:

```
#EXTM3U
#EXT-X-STREAM-INF:CLOSED-CAPTIONS=NONE,RESOLUTION=640x480,FRAME-RATE=25.000,CODECS="avc1.4d001f,mp4a.40.2",AVERAGE-BANDWIDTH=500000,BANDWIDTH=630000
tracks-v3a1/mono.m3u8
#EXT-X-STREAM-INF:CLOSED-CAPTIONS=NONE,RESOLUTION=768x576,FRAME-RATE=25.000,CODECS="avc1.4d001f,mp4a.40.2",AVERAGE-BANDWIDTH=610000,BANDWIDTH=770000
tracks-v2a1/mono.m3u8
#EXT-X-STREAM-INF:CLOSED-CAPTIONS=NONE,RESOLUTION=960x720,FRAME-RATE=25.000,CODECS="avc1.4d001f,mp4a.40.2",AVERAGE-BANDWIDTH=650000,BANDWIDTH=820000
tracks-v1a1/mono.m3u8
```

General standard HLS playback

When you have a simple live stream or file (one video track, one audio track) for playing, the URL for playback via HLS is simple:

```
http://FLUSSONIC-IP/STREAM_NAME/index.m3u8
```

where `FLUSSONIC-IP` stands for your Flussonic Media Server host + port address.

Flussonic Media Server will also accept `playlist.m3u8` in the end of the URL for backward compatibility with other servers.

With multilanguage or multi-bitrate content, there is a number of peculiarities in URL that are described further on this page.

HLS playback of fMP4 for H.265

Fragmented MP4 (fMP4) offers important benefits. First of all, it is the only way to **play HEVC video via the HLS**. Besides, the MP4 container is supported by any player, in contrast with MPEG. The fMP4 format can also used by DASH, so that only the manifest would be different from HLS while the MP4 encoding would be performed once for both protocols.

The following URL allows HLS playback as fMP4 chunks:

```
http://FLUSSONIC-IP/STREAM_NAME/index.fmp4.m3u8
```

where `FLUSSONIC-IP` stands for your Flussonic Media Server host + port address.

Playback via Low-Latency HLS

Flussonic allows playback via Apple Low-Latency HLS (LL-HLS).

Multilanguage HLS

If you want to play your multilanguage stream on iPhone you need to use the same `http://flussonic-ip/STREAM_NAME/index.m3u8`

But when you want to watch a multi-language stream using VLC or a set-top box, the `video` mode must be turned on.

URL for the player will be:

```
http://FLUSSONIC-IP/STREAM_NAME/video.m3u8
```

This is due to the Apple HLS requirement of a separate playlist with an audio-only option for each individual language. MPEG-TS uses another algorithm: all audio tracks are packed in the same container with the video, and it is up to the player which one to play. So, to make sure the video is viewable on iPhone, it must satisfy the requirements of Apple. At the same time, VLC and STBs, in violation of the HLS standard, expect the old version of MPEG-TS converted to HLS. This is why this trick with different URLs is needed.

Adding 'Audio only' for Apple devices

Apple requires that all your streams must include a version without video, only with audio.

They suppose that if a user is watching video via 3G and has moved to a zone with bad network conditions, it would be better to have audio only than video with buffering.

Flussonic allows you to enable audio-only mode in the following way:

```
stream example {
  input file://vod/bunny.mp4;
  add_audio_only;
}
```

> **Note**
>
> Such a configuration might make your `index.m3u8` URL unplayable on VLC or STB — in that case use `video.m3u8` option (described earlier on this page).

Separate playlists for STBs

If you have a multi-bitrate multilanguage content and want to play it on STB that doesn't support multi-bitrate HLS playlists, you can request from Flussonic Media Server separate playlists with one video track and all audio tracks like with the `mono` option:

```
http://FLUSSONIC-IP/STREAM_NAME/mono.m3u8
```

This playlist is not a variant (multi-bitrate) playlist, but it is a playlist with URLs to segments that contain the first video track and all available audio tracks.

If you want to deliver multi-language multi-bitrate to STB that doesn't understand Apple standard for multilanguage, use `video.m3u8` :

```
http://FLUSSONIC-IP/STREAM_NAME/video.m3u8
```

This is a variant playlist that will give you a list of non-variant playlists like `video1.m3u8` , `video2.m3u8` , etc.

SEPARATE AUDIO PLAYLISTS

Some Smart TVs (like Samsung TV) and browsers, supporting the *MSE (Media Source Extensions)* standard, cannot switch between the audio tracks if HLS stream has multiple audio tracks encoded (for instance, for different languages). As a result, players do not display these audio tracks, or a stream might not be played at all.

Flussonic can create a playlist with separate audio for such cases. To enable this feature, add `separate_audio=true` in the query string to request the HLS playlist for the player:

```
http://FLUSSONIC-IP/STREAM_NAME/index.m3u8?separate_audio=true
```

So the playlist will looks as follows:

```
#EXTM3U
#EXT-X-MEDIA:TYPE=AUDIO,GROUP-ID="aac",NAME="eng a1",DEFAULT=YES,AUTOSELECT=YES,LANGUAGE="eng",URI="tracks-a1/mono.m3u8"
#EXT-X-STREAM-INF:AUDIO="aac",CLOSED-CAPTIONS=NONE,RESOLUTION=320x240,FRAME-RATE=25.000,CODECS="avc1.420015,mp4a.40.2",AVERAGE-BANDWIDTH=240000,BANDWIDTH=310000
tracks-v1/mono.m3u8
```

It works for Live, VOD, and DVR playlists.

This is how to request such a playlist for DVR:

```
http://FLUSSONIC-IP/STREAM_NAME/archive-1643098810-30.m3u8?separate_audio=true
```

The output would look like so:

```
#EXTM3U
#EXT-X-MEDIA:TYPE=AUDIO,GROUP-ID="aac",NAME="eng a1",DEFAULT=YES,AUTOSELECT=YES,LANGUAGE="eng",URI="tracks-a1/index-1643098810-30.m3u8"
#EXT-X-STREAM-INF:AUDIO="aac",CLOSED-CAPTIONS=NONE,RESOLUTION=320x240,FRAME-RATE=25.000,CODECS="avc1.420015,mp4a.40.2",AVERAGE-BANDWIDTH=240000,BANDWIDTH=300000
tracks-v1/index-1643098810-30.m3u8
```

HLS DVR playback

DVR CATCHUP PLAYBACK

When your stream is already recorded on the server with our DVR you can play video via HLS when you know the beginning and the duration of a telecast, for example, from EPG.

Available URLs will be:

```
http://FLUSSONIC-IP/STREAM_NAME/archive-1659507585-3600.m3u8
```

This is a regular playlist that will be variant playlist if you have more than one language or more than one bitrate.

This playlist has the **VOD** (static) type. It contains a list of segments starting from UTC 1659507585 (2022, August, 3th, 06:19:45 GMT) and for one hour forward. No changes will be made to this playlist.

You may request the playlist of the **EVENT** (append-only) type if the end of the requested period is in the present or future. Add `event=true` in the above URL for that. Such a playlist will contain a list of segments starting from UTC 1659507585 (2022, August, 3th, 06:19:45 GMT) up to the current moment. New segments will be appended to this playlist, but all previously seen segments will be kept. If you use the same URL after some time, when the specified end of period will become the past, the playlist will get the **VOD** type.

The `mono` URL will give you list of segments that contain all tracks in MPEG-TS:

```
http://FLUSSONIC-IP/STREAM_NAME/mono-1659507585-3600.m3u8
```

More specific `videoN` playlist will give you a list of segments with an N'th video track and all audio tracks:

```
http://FLUSSONIC-IP/STREAM_NAME/video1-1659507585-3600.m3u8
```

and a variant video playlist with a list of `videoN` playlists that can be used for old STB and VLC:

```
http://FLUSSONIC-IP/STREAM_NAME/video-1659507585-3600.m3u8
```

> **Note**
>
> Please don't use `video` URL for browsers and modern clients. In most cases, we recommend to use the regular `archive` URL for getting playlist for video with multiple bitrates or multiple languages.

HLS EVENT PLAYLIST

You can access Event playlist with the URL:

```
http://FLUSSONIC-IP/STREAM_NAME/index-1659507585-now.m3u8
```

Event playlists are used when you want to allow the user to seek to any point within the event, e.g webinar, concert, current tv show. Such a playlist will contain a list of segments starting from UTC 1659507585 (2022, August, 3th, 06:19:45 GMT) up to the current moment. New segments will be appended to this playlist, but all previously seen segments will be kept.

Please notice that player will start playback from the live, not from the requested timecode, to access DVR data you should seek into the past.

REWINDING A PLAYLIST

Flussonic Media Server has a special playlist **"rewind-N.m3u8"** with a wide sliding window that allows to rewind and pause HLS streams for many hours.

```
http://FLUSSONIC-IP/STREAM_NAME/rewind-7200.m3u8
```

7200 is a duration of a HLS manifest in seconds, so your clients will be able to pause the stream up to two hours or rewind to the start of TV show without accessing catchup URLs.

DVR TIMESHIFT PLAYBACK

If your stream is being recorded on disk but you haven't configured a timeshifted stream for it, then you can play timeshifted video via HLS by using a propely constructed URL.

Relative timeshift:

- Playing `ago` seconds back in time: `http://FLUSSONIC-IP:PORT/STREAM_NAME/timeshift_rel-{ago}.m3u8`

Absolute timeshift:

- Playing `from` UTC moment in time: `http://FLUSSONIC-IP:PORT/STREAM_NAME/timeshift_abs-{from}.m3u8`

DVR FMP4 OVER HLS

Similar URL formats as for usual HLS are supported for playing fMP4 DVR over HLS:

- DVR catchup: `http://FLUSSONIC-IP:PORT/STREAM_NAME/archive-{from}-{depth}.fmp4.m3u8`
- DVR window: `http://FLUSSONIC-IP:PORT/STREAM_NAME/index-{from}-{duration}.fmp4.m3u8`
- Event playlist: `http://FLUSSONIC-IP:PORT/STREAM_NAME/archive-{from}-now.fmp4.m3u8`
- Rewind: `http://FLUSSONIC-IP:PORT/STREAM_NAME/rewind-{ago}.fmp4.m3u8`
- Absolute DVR timeshift: `http://FLUSSONIC-IP:PORT/STREAM_NAME/timeshift_abs-{from}.fmp4.m3u8`
- Relative DVR timeshift: `http://FLUSSONIC-IP:PORT/STREAM_NAME/timeshift_rel-{ago}.fmp4.m3u8`

Learn more about playing fMP4 over HLS in the section above.

Playing individual tracks

If a stream has several audio and video tracks, you can specify which tracks should be delivered. To do so, specify the track numbers by adding the parameter `filter.tracks` to the stream's URL.

Examples:

- Select the first audio and second video tracks:

```
http://FLUSSONIC-IP/STREAM_NAME/index.m3u8?filter.tracks=v2a1
```

- Select video only:

```
http://FLUSSONIC-IP/STREAM_NAME/index.m3u8?filter.tracks=v1
```

- Select the second video track and the first audio track to play the DVR archive section starting from UTC 1362504585 and with the duration of 3600 seconds:

```
http://FLUSSONIC-IP/STREAM_NAME/archive-1362504585-3600.m3u8?filter.tracks=v2a1
```

Sorting tracks in multibitrate playlist

If a multibitrate stream is used, all tracks are listed in the master playlist. By default, video tracks are sorted by bitrate in ascending order, i.e., playback starts from the video track with the lowest bitrate.

If you want to change the order of the tracks and start playing from another track, you can use the parameter `filter.tracks` for this purpose. Specify video and audio track numbers in this parameter in the desired order, and the tracks in the playlist will be sorted correspondingly. For example:

`http://FLUSSONIC-IP/STREAM_NAME/index.m3u8?filter.tracks=v3v2v1a2a1`

In this case default tracks will be `v3` and `a2`.

HLS playback with thumbnails

It is possible to add thumbnails into HLS playlist as special tags that a player can read. It works both for streams with DVR enabled and for VOD files.

To include thumbnails into the playlist, add `?thumbnails=` option to the stream or the VOD file URL.

Example for a DVR window:

`http://flussonic:80/ort/index-1644304617-60.m3u8?thumbnails=50`

Example for a VOD file:

`http://flussonic:80/vod/bunny.mp4/index.m3u8?thumbnails=100`

This value defines how many thumbnail links will be added to the thumbnail playlist to cover the duration of the DVR window or the VOD file correspondingly. The player will add the thumbnail links to the progress bar at regular intervals. The duration of the interval between thumbnails is the whole duration of the DVR window or the VOD file divided by this value.

If you specify a big number, then the player will use additional resources which may make the player or the browser stuck. Decreasing this parameter allows to limit the number of thumbnails to play and thus to reduce the player resources usage.

This option requires the parameters `thumbnails enabled=ondemand` and `size` (thumbnail size) included in the stream or VOD location settings. For example, `thumbnails enabled=ondemand size=320x240;`. You can specify multiple sizes separated by spaces, for example, `size=320x250 size=640x480`. In this case, several thumbnails tracks will be included into the playlist. Each thumbnail in the playlist will be resized proportionally to fit the specified size.

For more information, please refer to Streaming API schema.

3.8.3 LL-HLS playback

Flussonic allows playback via Apple Low-Latency HLS (LL-HLS) — a streaming protocol that derives from HLS and overcomes its high latency.

LL-HLS supports the same codecs as HLS (H.264, AAC, MP3) and also HEVC (H.265) and AV1. The container can be MPEG-TS or fMP4 (fragmented MP4). *Flussonic* uses fMP4 and CMAF to package streams for LL-HLS delivery. *Flussonic* creates fMP4 chunks in accordance with the CMAF standard.

Before using Low-Latency HLS, remember that network and server load will increase because Low-Latency HLS divides HLS segments into smaller segments (also called chunks).

LL-HLS playback URL

To play a stream via Apple Low-Latency HLS, copy the **CMAF** URL from the **Output** tab of the stream's settings and paste it to a player. The player THEOplayer fully supports LL-HLS playback.

***CMAF** is a standard that is used to create MP4 fragments compliant with the Low-Latency HLS specification.*

The URL is like that:

```
http://FLUSSONIC-IP/example_stream/index.ll.m3u8
```

You also can playback LL-HLS from embed.html player

```
http://FLUSSONIC-IP/example_stream/embed.html?realtime=true&proto=ll-hls
```

If the stream includes some tracks that you don't want in the LL-HLS output, use `filter.tracks` parameter in the URL. It will help you filter only the tracks that are required on the end-user's device (for example, you may not want to deliver 360p to TVs or 4K to mobile devices). Learn more about filtering tracks at Filtering tracks for playback.

Optional settings for LL-HLS

LL-HLS is enabled in *Flussonic* by default and **does not require any additional settings**. We have adjusted the LL-HLS playback parameters to achieve an optimal balance between latency and resource consumption. However, you can change the settings if any issues arise with the default values (for example, the video takes a long time to start, the server is overloaded at a small number of viewers, frequent buffering, the latency is higher than expected, etc.). Please contact our support team to get assistance in selecting the best LL-HLS settings for your objectives.

You can set the following parameters in the configuration file (**if necessary**):

- segment_duration is the duration of an HLS segment.
- segment_count is the number of segments in an HLS playlist.
- chunk_duration is the duration of a CMAF Chunk or HLS Partial Segment of an HLS segment. The default value is `200`.

Example:

```
stream example_stream {
  input fake://fake;
  segment_duration 4;
  segment_count 4;
  chunk_duration 500;
}
```

> **Note**
>
> You can also set `segment_duration` and `segment_count` in the UI on the **Output** tab in the stream settings.

3.8.4 WebRTC playback

You can play any stream via WebRTC including streams published by WHIP as described in WebRTC Publishing or any other stream configured on your Flussonic server. You can use our `embed.html` player for the playback, or any player with WebRTC support, or your own application.

The playback is carried out via the WHEP standard. For more details about WebRTC, WHIP and WHEP, see Using WebRTC protocol.

URLs for playback via WebRTC

To play the stream, use our embed.html player opening the following URL in the browser:

```
http://FLUSSONIC-IP/STREAM_NAME/embed.html?proto=webrtc
```

where:

- `FLUSSONIC-IP` is an IP address of your *Flussonic* server
- `STREAM_NAME` is the name of your WebRTC stream

You can also use another player that supports WebRTC or develop your own application; in that case use the following URL:

`http://FLUSSONIC-IP:PORT/STREAM_NAME/whep`

See Streaming API reference.

The code must be run on the client side that plays video from that URL. Learn more at Recommendations on developing the client application.

Load balancing with WHEP playback

Since WHEP is based on HTTP POST requests, you can use our load balancer to distribute play requests between servers in a cluster. The balancer will redirect POST requests to servers in the cluster using the 307 HTTP redirect code.

Adaptive streaming

In order to adjust the bitrate of the played stream to the bandwidth of the user's channel, you can configure adaptive streaming, i.e. playback with adaptive bitrate (ABR). This way, each of your users will be able to receive a video of the highest possible quality at their channel capabilities.

3.8.5 Adaptive bitrate streaming over WebRTC

When streaming user-generated content, you will probably face the necessity to provide each viewer with the highest video quality their network connection can afford. To address this requirement, the WebRTC protocol provides the *adaptive streaming* technology which is successfully implemented in *Flussonic*.

Adaptive streaming is based on ***Adaptive Bitrate (ABR)*** algorithm designed to deliver video efficiently to a wide range of devices. In *adaptive bitrate* streaming, multiple bitrate (MBR) renditions of the same source—*tracks*—are created using the transcoder. The channel between server and each individual client player is then established to send one of the tracks; in the ABR mode, the server decides which track to send depending on the user's current network speed. When necessary, manual track selection can be provided in the client player.

Flussonic selects a track with bitrate and quality suitable for the user based on the following indicators received from the user's browser:

- *NACK* (Negative ACKnowledgement) packet loss indicator. This is the number of lost packets per unit of time.
- *REMB* (Receiver Estimated Maximum Bitrate) or *TWCC* (Transport-wide Congestion Control) packet rate indicator. This is the time it takes for a packet to be delivered from the server to the client player. See Using REMB or TWCC for ABR below for details.

Configuring ABR

The ABR option is enabled for WebRTC by default. This means that players will work in `auto` mode until a user chooses the resolution manually. To enable the `auto` mode again, select it in the player's settings.

Make sure to configure the transcoder to define the tracks for ABR switching, for example:

```
stream webrtc-abr {
  input fake://;
  webrtc_abr;
  transcoder vb=1000k size=1920x1080 bf=0 vb=300 size=320x240 bf=0 ab=64k acodec=opus;
}
```

Using REMB or TWCC for ABR

When playing streams via WebRTC, *Flussonic* uses RTP protocol for sending video and audio frames. This protocol provides two mechanisms of measuring available bandwidth. *Flussonic* can use one of those mechanisms in ABR algorithm to decide if it is possible to switch bitrate to a higher value.

REMB

You can choose the mechanism based on **REMB (Receiver Estimated Maximum Bitrate)** message reported by the client. The bitrate of sent video will not exceed the one sent by the client in the REMB. However, if REMB grows, *Flussonic* can switch to a track with a higher value. Learn more about REMB.

This is a simple mechanism, however it has some drawbacks:

- After temporary packet loss (for example, due to network connection failure), REMB falls dramatically and then increases too slowly (for 5-15 minutes). During this time *Flussonic* cannot switch to a track with better quality for a long time although the client is able to play it.
- *Flussonic* cannot control this value because it is calculated on the client's side.
- This mechanism is marked as deprecated and probably will not be developed.

To enable REMB mechanism, set `bitrate_prober=false` in the `webrtc_abr` directive.

TWCC

By default, *Flussonic* uses the **TWCC (Transport-wide Congestion Control)** mechanism available as RTP extension. Learn more about TWCC.

In this case, *Flussonic* adds to each sent packet an RTP header extension that contains the extension ID and the packet sequence number. The client sends back an RTCP feedback message containing the arrival times and sequence numbers of the packets received on a connection. Thus, *Flussonic* knows the sending time and receiving time of each packet and can calculate the difference between them. Also, *Flussonic* knows the size of each packet, so it can calculate the bitrate the packets were actually sent with.

To estimate maximum possible bitrate, *Flussonic* sends groups of so called `probe` packets at regular intervals. These packets are sent with a bitrate higher than the current one. When the packets are received, *Flussonic* calculates their actual bitrate as described above. If after some iteration the calculated bitrate exceeds the bitrate of the next (higher quality) track at 10% and the packet loss (NACK) conditions are satisfactory, *Flussonic* switches to the next track.

This mechanism provides more control and flexibility because most of its logic works on the sender side.

You can set the following parameters of TWCC mechanism in the `webrtc_abr` directive:

- `bitrate_prober=true` – switch to using TWCC
- `bitrate_probing_interval` – the time interval of sending `probe` packets, in seconds.

For example:

```
webrtc_abr bitrate_prober=true bitrate_probing_interval=2;
```

3.8.6 DASH Playback

Flussonic Media Server supports playing video via the DASH protocol.

The supported codecs are: H264, H265, AAC, MP3, and AC-3.

Flussonic Media Server supports access via MPEG-DASH to live streams, VOD files, and DVR (catchup and timeshift).

If an incoming stream has DVB subtitles, then Flussonic can pass them to an output MPEG-DASH stream if you configure Flussonic for that. Subtitles are saved in DVR archive if a stream is recorded to the archive.

DASH uses a manifest file for transmitting information about a requested stream. To keep it simple, we call it 'playlist' here.

On this page:

Simple video playback via DASH

When you have a live stream or file (one video track, one audio track) for playing, the URL for playback via DASH is simple:

```
http://FLUSSONIC-IP/STREAMNAME/index.mpd
```

where `FLUSSONIC-IP` is a placeholder for your Flussonic Media Server host + port address.

Selecting tracks for playback

If a stream has several audio, video, or subtitle tracks, you can specify the tracks for playback. To do this, use the filter.tracks parameter.

See the following examples:

- Select the first audio and second video tracks:

```
http://FLUSSONIC-IP/STREAMNAME/index.mpd?filter.tracks=v2a1
```

- Select video only:

```
http://FLUSSONIC-IP/STREAMNAME/index.mpd?filter.tracks=v1
```

- Select the second video track and the first audio track for the DVR archive starting from UTC 1362504585 with the duration of 3600 seconds:

```
http://FLUSSONIC-IP/STREAMNAME/archive-1362504585-3600.mpd?filter.tracks=v2a1
```

DVR catch up playback via DASH

When your stream is already recorded on a server with our DVR, you can play video via DASH if you know the beginning and ending time of telecast, for example, from EPG.

URLs for playing video from archives will be like this:

```
http://FLUSSONIC-IP/STREAMNAME/archive-1362504585-3600.mpd
```

Such a URL will give a list of segments starting from UTC 1362504585 (2013, March, 5th, 17:29:45 GMT) and for one hour forward (3600 seconds).

If you have more than one language or more than one bitrate, you will get an adaptive stream with the possibility to select the audio track.

Rewinding DASH videos

Flussonic Media Server has a special playlist **"rewind-N.mpd"** with a wide sliding window that allows you to rewind DASH streams and pause them for many hours.

```
http://FLUSSONIC-IP/STREAMNAME/rewind-7200.mpd
```

Here **7200** is the duration of the DASH manifest in seconds, so your clients will be able to pause the stream for up to 2 hours or rewind to the start of a TV show without using catchup URLs.

Also, there is a playlist in which you can receive a live stream and rewind up to a specified time: **"archive-N-now.mpd"**, where N — is a Unix timestamp of the moment to which users can rewind the stream.

```
http://FLUSSONIC-IP/STREAMNAME/archive-1362504585-now.mpd
```

DVR timeshift playback

Here we descibe one more way to play an archive via DASH with the option to rewind up to the specified time. If you haven't configured a timeshifted stream, you can still play timeshifted video via DASH by using a propely constructed URL.

Here goes the example of an URL for absolute timeshift:

```
http://FLUSSONIC-IP/STREAMNAME/timeshift_abs-1584435600.mpd
```

where 1584435600 is 03/17/2020 @ 9:00am (UTC)

The player will play live at the start and allow rewinding up to 1584435600.

DASH manifests for playing DVR on TVs with WebOS

Flussonic can create DASH manifest of two types: with multiple periods and with a single period.

Initially, Flussonic designed its DASH manifest for DVR playback with the view to usage in CDN. The manifest with multiple periods was suitable for this purpose.

However, such a manifest is incompatible with a wide range of devices and TV sets used by consumers in many countries, such as US. These include LG TVs on WebOS and others.

For devices that cannot play DASH with multi-period timeline, we designed a single-period manifest enabling you to play DASH on that devices.

Add `period=mono` to the URL as follows:

```
http://FLUSSONIC-IP/STREAMNAME/archive-TIME-DURATION.mpd?period=mono
```

or

```
http://FLUSSONIC-IP/STREAMNAME/archive-TIME-now.mpd?period=mono
```

Note. The single-period manifest for live with the option to view the recorded archive (archive-TIME-now.mpd?period=mono) is sensitive to the quality of the input stream source — it is necessary that there are no gaps in the live stream.

Turning on DVB compliance of a DASH manifest

If you use a validator for DASH and you turn on checking for DVB compliance in the validator, you'll need to make sure that your DASH manifests are compliant with the DVB profile.

In order to get a DVB-compliant manifest, add the option `dvb=1` to the stream's URL:

```
http://FLUSSONIC-IP/STREAMNAME/index.mpd?dvb=1
```

Playback with subtitles

Flussonic supports passing both TTML and WebVTT subtitles into DASH streams. This allows showing subtitles on a wider range of devices and set-top boxes.

Choosing subtitles format for DASH playback

As two formats of subtitles are included in a DASH manifest, you can choose one of them when playing an output stream:

```
https://FLUSSONIC-IP/STREAMNAME/index.mpd?text=wvtt
```

or (TTML is the default format)

```
https://FLUSSONIC-IP/STREAMNAME/index.mpd?text=ttml
```

The option `text` can be used also with URLs like:

```
http://FLUSSONIC-IP/STREAMNAME/rewind-7200.mpd?text=wvtt
```

DASH playback with thumbnails

It is possible to add thumbnails into DASH playlist as special tags that a player can read. It works both for streams with DVR enabled and for VOD files.

To include thumbnails into the playlist, add `?thumbnails=` option to the stream or the VOD file URL.

Example for a DVR window:

`http:// flussonic:80/ort/archive-1643013512-now.mpd?thumbnails=50`

Example for a VOD file:

`http://flussonic:80/vod/bunny.mp4/Manifest.mpd?thumbnails=100`

This value defines how many thumbnail links will be added to the thumbnail playlist to cover the duration of the DVR window or the VOD file correspondingly. The player will add the thumbnail links to the progress bar at regular intervals. The duration of the interval between thumbnails is the whole duration of the DVR window or the VOD file divided by this value.

If you specify a big number, then the player will use additional resources which may make the player or the browser stuck. Decreasing this parameter allows to limit the number of thumbnails to play and thus to reduce the player resources usage.

This option requires the parameters `thumbnails enabled=ondemand` and `size` (thumbnail size) included in the stream or VOD location settings. For example, `thumbnails enabled=ondemand size=320x240;`. You can specify multiple sizes separated by spaces, for example, `size=320x250 size=640x480`. In this case, several thumbnails tracks will be included into the playlist. Each thumbnail in the playlist will be resized proportionally to fit the specified size.

For more information, please refer to Streaming API schema.

3.8.7 SRT playback

Flussonic supports playing SRT streams. Read more about SRT at Using SRT protocol.

SRT port is usually configured *per stream*, i.e. one SRT stream per port. Nevertheless, *Flussonic* provides you with the way to set up a single *global* SRT port for *multiple streams*. For example, if you use SRT for restreaming only, it may come in handy. The playback URL depends on the way you choose to specify the port, read more

One SRT stream per port

To configure an SRT port to play a single SRT stream, use `srt_play` configuration in the stream settings like so:

```
stream example_stream {
  input fake://fake;
  srt_play {
  port 9998;
  }
}
```

To play the stream, use the following URL:

- `srt://FLUSSONIC-IP:SRT_PORT`

where:

- `FLUSSONIC-IP` is an IP address of your *Flussonic* server.
- `SRT_PORT` is an SRT port, specified for playback.

Thus, for our `example_stream` the link will look as follows: `srt://localhost:9998`. Using the `srt_play` you `allow to play` the SRT stream over a specified port.

You can also define an SRT port for a particular stream not only for a playback, but also for publishing (see Publishing SRT stream to learn more about publishing SRT streams to *Flussonic*). To configure a single SRT port to publish *and* play a stream, use `srt PORT_NUMBER` option in the stream settings:

```
stream example_stream {
  input publish://;
  srt 9998;
}
```

In the example above we publish some SRT stream to `example_stream` over port `9998` and we can also play this stream over the same port, using the following URL format:

- `srt://FLUSSONIC-IP:SRT_PORT?streamid=#!::m=request`

where:

- `streamid` is a string formatted as described here.
- `m=request` is a playback mode.

`streamid` is used here to specify the mode `m=` to play the stream `example_stream` as it is not obvious now whether we are going to publish the stream or to play it.

So the URL for our stream is `srt://localhost:9998?streamid=#!::m=request`.

Besides using one SRT port for one stream to *allow playback for a particular stream*, *Flussonic* provides you with the way to use one port for multiple SRT streams.

One SRT port to play multiple streams

To enable one SRT port for playback only for any number of streams, use `srt_play` as a global setting:

```
srt_play {
  port 9998;
}
stream example_stream {
  input fake://fake;
}
```

To play `example_stream` over the `9998` SRT port, use the following URL:

- `srt://FLUSSONIC-IP:SRT_PORT?streamid=#!::r=STREAM_NAME`

where:

- `streamid` is a string formatted as described here.
- `r=STREAM_NAME` is a stream name.

So for the above example the URL looks as follows: `srt://localhost:9998?streamid=#!::r=example_stream`.

Let us consider one more example. Say you publish an SRT stream to *Flussonic* over a specified publishing port, and then you need to play it along with other streams. So the example configuration looks as follows:

```
srt_play {
  port 9998;
}
stream example_stream {
  input fake://fake;
}
stream another_stream {
  input publish://;
  srt_publish {
    port 8888;
  }
}
```

The URL for `another_stream` is different from the URL for the `example_stream` stream, and looks as follows:

- `srt://FLUSSONIC-IP:SRT_PORT?streamid=#!::r=STREAM_NAME,m=request`

where:

- `m=request` is a playback mode.
- `r=STREAM_NAME` is a stream name.

So the resulting URL for our example: `srt://localhost:9998?streamid=#!::r=another_stream,m=request`.

The summary is tabulated in the table below for your convenience:

Playback URL	Configuration	Description	URL example
`srt://FLUSSONIC-IP:SRT_PORT`	stream example_stream { input fake://fake; srt_play { port 9988; } }	Allows playing only one stream per port. Supported by most players.	`srt://localhost:9988`
`srt://FLUSSONIC-IP:SRT_PORT?streamid=#!::m=request`	stream example_stream	Allows publishing *and* playing a stream over the same SRT port.	`srt://localhost:8888?streamid=#!::m=request`
`srt://FLUSSONIC-IP:SRT_PORT?streamid=#!::r=STREAM_NAME`	srt_play { port 9999; } stream example_stream { input fake://fake; }	Allows playing several streams over the same SRT port.	`srt://localhost:9999?streamid=#!::r=example_stream`
`srt://FLUSSONIC-IP:SRT_PORT?streamid=#!::r=STREAM_NAME,m=request`	srt_play { port 9988; } stream example_stream { input fake://fake; } stream another_stream { input publish://; srt_publish { port 8888; } }	Having a specified publishing port for the stream allows playing the stream over a globally defined SRT port.	`srt://localhost:9988?streamid=#!::r=another_stream,m=request`

Parameters for an SRT playback

Flussonic allows you to manage the playback by setting the parameters.

Example with `passphrase`:

```
stream example_stream {
  input fake://fake;
  srt_play {
    passphrase 0987654321;
    port 9998;
  }
}
```

The URL will look like this:

- `srt://FLUSSONIC-IP:9998?passphrase=0987654321&streamid=#!::m=request`

3.8.8 MSS playback

Flussonic Media Server supports playing video via the MSS protocol. The stream is available at the URL:

`http://FLUSSONIC-IP/STREAMNAME.isml/manifest`

Selecting tracks

Selecting tracks is useful to play video on client devices that do not support, for example, the multi-language MSS manifest.

If a stream has several audio, video, and subtitle tracks, you can specify which tracks to play. To do this, specify track numbers by adding the `filter.tracks` parameter to the stream URL.

- `http://FLUSSONIC-IP/STREAMNAME.isml/manifest?filter.tracks=v1a1` — select the first audio and second video tracks.
- `http://FLUSSONIC-IP/STREAMNAME.isml/manifest?filter.tracks=a1` — select audio only.
- `http://FLUSSONIC-IP/STREAMNAME.isml/manifest?filter.tracks=v1` — select video only.
- `http://FLUSSONIC-IP/STREAMNAME.isml/manifest?filter.tracks=a1t2` — select the first audio and second subtitle tracks.
- `http://FLUSSONIC-IP/STREAMNAME.isml/manifest?filter.tracks=v1t1t2t3` — select the first video track and three tracks with subtitles.

MSS DVR catch up playback

You can request a fragment of an archive as a file by using the following URL:

`http://FLUSSONIC-IP:PORT/STREAM_NAME(archive=1651829645-120).isml/manifest`

See API reference.

MSS event playback

Accessing the archive starting from the present moment (that is, live) and with the possibility to rewind back to the specified time (1651829645 in the example):

`http://FLUSSONIC-IP:PORT/STREAM_NAME(archive=1651829645-now).isml/manifest`

See API reference.

MSS rewinding

Playlist with a wide sliding window that allows you to rewind MSS streams and pause them for many hours:

`http://FLUSSONIC-IP:PORT/STREAM_NAME(rewind=7200).isml/manifest`

MSS absolute timeshift

URL for MSS playback with absoulte timeshift:

`http://FLUSSONIC-IP:PORT/STREAM_NAME(timeshift_abs=1651829645).isml/manifest`

MSS relative timeshift

URL for MSS playback with relative timeshift:

`http://FLUSSONIC-IP:PORT/STREAM_NAME(timeshift_rel=600).isml/manifest`

3.8.9 HTML5 (MSE-LD) Low Latency Playback

For a long time the Flash player was the best and the only way to deliver video to web pages with relatively low latency (delay). Low latency is required for webinars, broadcasting sports for bets, video surveillance, or some kinds of remote control.

Right now Flash is scheduled for graceful removal from modern browsers, so the protocol WebRTC was added to browsers, but it has limited support for audio and video codecs (not all flavors of H.264 are supported, no AAC support).

Flussonic offers a new way to solve this problem and offers the player that allows watching video with really low latency through the browser's built-in HTML5 and the Media Source Extensions (MSE) mechanism.

Low latency playback

Let's play the stream from your Flussonic in the browser via the MSE mechanism. Open the following URL in the browser, replacing the server domain and stream name with your own:

`http://flussonic-ip/STREAMNAME/embed.html?realtime=true`

If everything is OK (good codecs, working stream, working websockets), you will instantly get video with the delay of about one second.

Under the hood

We use the MSE mechanism to deliver and play frames, so the supported video/audio codecs will be the same as in your browser. Usually H264 and AAC are supported, the rest is not supposed to work.

You don't need anything except HTTP or HTTPS to run this, so it may be a good way to play video in restricted environments.

You can also use our player inside your application without using iframe. Read about how to embed our MSE JavaScript player into your applications.

3.8.10 JPEG thumbnails

Video is a stream of pictures. Sometimes you need to extract these pictures and handle them separately from each other. Such separate pictures are called thumbnails or screenshots.

Flussonic Media Server can create thumbnails of a video stream. They allow you to:

- show an instant preview of a live stream on a web page to know what is happening there right now,
- take a look at the quality of a stream,
- freeze a point in time to use the screenshot somewhere else,
- make a fast search in the DVR archive for some fragment of video identifiable by the screenshot,
- create a wall of screenshots to quickly look at a whole day of recording,
- do whatever else with small static images extracted from a large video stream.

Flussonic Media Server can create thumbnails in two different ways:

- Extracting video frames as JPEG images. This is a resource-intensive operation. Flussonic can save JPEG thumbnails in the DVR archive. Learn more on this page.
- Creating resource-saving MP4 video thumbnails. In H.264 streams with keyframes, all compressed images are available without resource-intensive processing. Flussonic Media Server takes the first keyframe from each segment and displays it as an MP4 video consisting of one frame. **Learn more** in Video thumbnails section.

About JPEG thumbnails

Flussonic Media Server does a rather CPU-intensive job: it takes the first keyframe of each segment, decodes it to raw video, and encodes back to a JPEG image. This seems rather simple, but when you have, say, 300 streams, this process can take a lot of CPU time.

Flussonic allows some optimization here – by changing a segment's duration you can change the total number of JPEG thumbnails. The fact that Flussonic Media Server takes only the first keyframe of a segment means that if you configure the segment duration of 3 seconds, you'll have 20 JPEGs per minute. If you configure the segment duration to be 6 seconds, you'll have 10 JPEGs per minute. If you take a stream from an IP camera, you may have 60 keyframes per minute, but Flussonic Media Server will create a smaller number of JPEGs.

> **Note**
>
> If you enable DVR on a stream, all of the generated JPEGs will be written to the disk.

It is possible to optimize the CPU usage by accessing thumbnails by their URL. Usually it is suitable for IP cameras because IP camera maintain fresh JPEG screenshot for the currently shown video. In this case Flussonic Media Server will download a JPEG image each time video segment starts.

Configuring JPEG thumbnail generation

To create JPEG thumbnails, Flussonic Media Server uses a built-in package.

Add the `thumbnails` option to the stream settings:

```
stream example {
  input fake://fake;
  thumbnails;
}
```

This will start the `flussonic-thumbnailer` process. It may be rather resource-hungry — this is the nature of video and image compression.

All the settings for JPEG thumbnails can be specified through the administrator's panel in a stream's settings on the **Output** tab (**Media** > select a stream > **Output**). Select the **enabled** option in the **Thumbnails** section.

Configuring JPEG thumbnails downloading from camera

To reduce CPU usage on thumbnail generation, you can specify the URL where Flussonic Media Server can get thumbnails. Many cameras have a special URL for screenshots:

```
stream example {
  input rtsp://localhost:554/source;
  thumbnails url=http://examplehost:5000/snapshot;
}
```

You can try to find the screenshot URL in your camera's documentation, or look for that information on the web.

All the settings for JPEG thumbnails can be specified through the administrator's panel in a stream's settings on the **Output** tab (**Media** > select a stream > **Output**). Select the **enabled** option in the **Thumbnails** section and specify **Thumbnail URL**.

Getting JPEGs from live streams

After you have enabled thumbnails in Flussonic Media Server, you need to access them.

The URL for getting thumbnails is as follows:

`http://FLUSSONIC-IP:80/STREAM_NAME/preview.jpg` — a stream's last screenshot.

`http://FLUSSONIC-IP:80/STREAM_NAME/preview.mjpeg` — an MJPEG screenshot stream.

We strongly recommend that you never use the MJPEG stream because it is an uncontrollable way of streaming video with a very high bitrate. You can end up with an MJPEG stream with 50% of the original bitrate, streaming at 0.1fps. But if you still need it, you can use it.

Getting JPEGs from DVR by specific time

Screenshots are automatically saved to the archive if DVR is turned on for the stream. They can be obtained using the HTTP API.

The best way (in terms of resources) of getting JPEG screenshots is to specify an approximate UTC time as part of the URL. Flussonic will return the URL corresponding to the nearest keyframe (an actual screenshot).

`http://FLUSSONIC-IP:80/STREAM_NAME/1652936935.jpg`

The old human-readable GMT format is also supported for compatibility:

`http://FLUSSONIC-IP:80/STREAM_NAME/2022/05/19/05/08/11.jpg`

You then use this URL to access the screenshot.

See Streaming API reference.

Getting JPEGs from DVR by UTC time range

Warning

This method is resource-intensive, we don't recommend using it. A better way is to use an approximate UTC or GMT time. Learn more in the section **Getting JPEGs from DVR by specific time**.

First, you need to identify a time range for which you want to get DVR. For example, right now it is 2017 April 21, 13:10 GMT, so it is 1492780200 UTC. If you want to get thumbnails for the last hour, you need to request the following URL:

```
curl 'http://FLUSSONIC-IP:80/STREAM_NAME/recording_status.json?from=1492776600&to=1492780200&request=brief_thumbnails'
```

By default, Flussonic does not include the list of timestamps in the response. To get them, we need to add `request=brief_thumbnails` to the query string.

The response will look like this:


```
[{"stream":"clock","ranges":[{"duration":3642,"from":1492776599}],"brief_thumbnails":[1492776599,1492776605,
1492776617,1492776629,1492776641,1492776653,1492776665,1492776677,1492776689,1492776701,1492776713,1492776725,
....]}]
```


Here you get a very long list of timestamps that you need to convert to paths to screenshots. For example, the timestamp `1492776605` will be converted to `http://FLUSSONIC-IP:80/STREAM_NAME/2017/04/21/12/10/05.jpg`.

You will only get a list of timestamps, you will need to get the thumbnails themselves by requesting them individually.

On-demand JPEG thumbnail generation

Sometimes it is very expensive to store all JPEG images on the disk, so you can ask *Flussonic Media Server* to generate JPEGs on demand. In this case, add the parameter `thumbnails enabled=ondemand` to the stream configuration. For example:

```
stream channel {
  input fake://fake;
  thumbnails enabled=ondemand;
  dvr /storage;
}
```

You can also configure JPEG thumbnails on-demand generation through the administrator's panel in a stream's settings on the **Output** tab (**Media** > select a stream > **Output**). Select the **On demand** option in the **Thumbnails** section.

Request a URL with a certain time specified in UTC format:

`http://FLUSSONIC-IP:80/STREAM_NAME/1643797938-preview.jpg`

The old human-readable datetime format is also supported for compatibility:

`http://FLUSSONIC-IP:80/STREAM_NAME/2022/02/21/12/10/05-preview.jpg`

Flussonic Media Server will find a segment, take the first keyframe after the specified moment and generate a JPEG image from it.

If no keyframes are found in the current segment, Flussonic will take the first keyframe of the next segment, create its URL, return 302 redirect, and the browser will make the second request with the new URL and get the JPEG image of the found keyframe. Such redirection ensures storing only one unique response for each URL in the cache. So, if no keyframes are found for a requested time, the already saved correct URL can be taken from the cache after the redirection. And two requests to neighbouring seconds without keyframes will result in downloading only one JPEG image.

This method might lead to unpredictable CPU usage, so it is not recommended.

With the JPEG thumbnailer enabled, you will have smooth and moderate CPU usage, without spikes in load. With on-demand JPEG generation, you may have low CPU usage overall, but during prime time you may get spikes, and your server may become unstable.

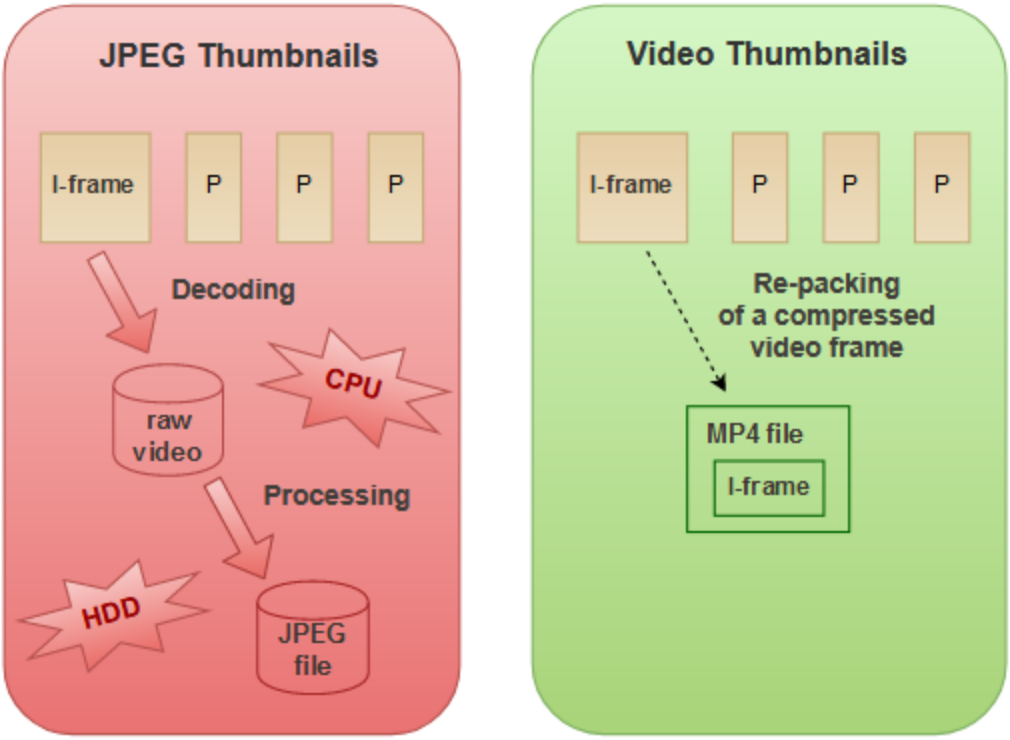
See Streaming API reference.

3.8.11 MP4 video thumbnails

About video thumbnails

Why use video thumbnails

If you want to minimize system overhead, use our *JPEG-less video thumbnails*. Video thumbnails eliminate the drawbacks of JPEG thumbnails: increased CPU usage and increased disk space usage. We can make screenshots available without creating JPEG files. This way, the overhead is almost 0%.



> **Note**
>
> JPEG-less video thumbnails work with H.264 video only.

How video thumbnails work

JPEG-less video thumbnails are essentially fragments of H.264 video each containing one frame. If you have an H.264 stream with keyframes, then you already have compressed images in it — keyframes.

Because Flussonic can obtain these images from a video on the fly, there's no need to store them separately. There is no need to produce them either — so CPU time is saved immensely. All you need is to access ready keyframes.

Flussonic takes the first keyframe from each segment and creates an MP4 file from it. This file is sent to the browser, where it is then shown as a picture. This is a *video thumbnail*.

An example of code for showing a video thumbnail in a browser:

```
<video src="http://flussonic:80/clock/preview.mp4" style="width: 640px; height: 480px;" autoplay />
```

When you insert such a tag to your HTML page, the page will show the thumbnail. It is also possible to access video thumbnails on mobile browsers, or in DVR and VOD player.

> **Note**
>
> Don't forget to remove the `thumbnails` option from a stream's configuration, it is required only for JPEG.

HOW TO ACCESS VIDEO THUMBNAILS

To view a thumbnail in a web browser, you should request it by using a special URL. You can use this URL in the HTML tag `<video>`.

In general, URLs look like this:

`http://<domain>/<stream name or path to file>/preview.mp4`

Such URLs differ for live, VoD, and DVR thumbnails depending on the place in a video that you want to show on a thumbnail. But they always contain a `preview.mp4` file. There are 4 situations:

1. Live video thumbnails (of a live stream) – in this case, the very latest keyframe is used to create a thumbnail.

   `http://FLUSSONIC-IP:80/STREAM_NAME/preview.mp4`
2. Video thumbnails of a DVR archive. You can obtain them by specifying date and time as part of URL. See details later on this page.

   `http://FLUSSONIC-IP:80/STREAM_NAME/1654242430-preview.mp4`
3. Video thumbnails of a file (VoD). You can obtain them by specifying time passed from the befinning of the file (HH-MM-SS). See details later on this page.

   `http://FLUSSONIC-IP:80/vod/bunny.mp4/preview-01-23-55.mp4`
4. A video thumbnail of the first keyframe in a file. The drawback is that the first keyframe sometimes contains a black picture.

   `http://FLUSSONIC-IP:80/vod/bunny.mp4/preview.mp4`

Video thumbnails of files in VoD

You can get Flussonic's video thumbnails of any place in a file, not only the beginning. You specify the time and Flussonic shows the nearest keyframe. Technically, Flussonic takes only one (the first) keyframe of each segment.

To access a thumbnail, add time to the URL and put this URL in a `<video>` tag that you insert into a web page. Use the following syntax for the URL:

`http://<domain>/<path>/<filename>.mp4/preview-<Hour-Minute-Second>.mp4`

To show the very first keyframe, use the URL without time:

`http://<domain>/<path>/<filename>.mp4/preview.mp4`

This example shows a thumbnail of the video at 02:24:45:

```
<video src="http://flussonic:80/vod/bunny.mp4/preview-02-24-45.mp4" style="width: 640px; height: 480px;" autoplay />
```

Flussonic redirects to the URL that has a calculated keyframe number instead of the specified time.

Video thumbnails of DVR

It is possible to access video thumbnails from DVR in the same manner as with JPEG thumbnails: for the UTC timestamp 1654242430 (which corresponds to 2022 June 3, 07:47:10 GMT) you need to request:

`http://FLUSSONIC-IP:80/STREAM_NAME/1654242430-preview.mp4`

The old human-readable GMT format is also supported for compatibility:

`http://FLUSSONIC-IP:80/2022/06/03/07/47/10-preview.mp4`

But we have implemented a more convenient way to access these thumbnails. If you know that somewhere in 10 minutes after a point in time you have recorded video, you can request inexistent URL (with approximate time). *Flussonic* will find the existing keyframe in that period and return it. This approach will save your cache: the browser will make two requests, but only the existing keyframe will be saved to the browser cache, it prevents it from spoiling.

For example, you can request an inexistent URL for a moment that is 10 minutes earlier than in the previous example:

`http://FLUSSONIC-IP:80/STREAM_NAME/1654241830-preview.mp4`

For existing thumbnail Flussonic Media Server will return the header `X-Thumbnail-Utc: 1654242430`, you can use it to understand real URL of thumbnails as browser will not tell you about the redirect.

Video thumbnails on mobile devices

Just use the below code to show video thumbnails on mobile devices:

```
<video src="http://flussonic:80/clock/preview.mp4" autoplay playsinline /></video>
```

The `playsinline` parameter insures that the preview is displayed correctly (for example, does not overlap the elements placed over it).

3.8.12 Overlaying logos

Flussonic offers two ways to overlay a logo to a video stream with *Flussonic Media Server*:

- Using the embed.html web player. The player adds a transparent layer with your image on top of the video stream. This method doesn't generate additional load on the server and works well if you want to embed a video on a website.
- Using the transcoder. Transcoder burns the logo image into the video stream so the logo can't be hidden or removed from the video. It's a computationally expensive process. This method is suitable for video streams transmitted to set-top boxes.

Adding a logo to a video stream using the embed.html web player

You can add a logo to a video stream using the embed.html web player in two ways:

- In the Flussonic UI
- In the Flussonic configuration file

You can see the logo in the live stream and the DVR player.

IN THE FLUSSONIC UI

1. In the **Media** > **Streams**, open the stream settings by clicking on the stream name.
2. In the **Output** tab, find the **Logo** section.
3. Upload the logo by clicking **Select** > **Add New** and selecting the logo image. You can upload several images.

> **Warning**
>
> You should add a logo only with .png extension.

1. Optional: Adjust the image size and its position in the video using these parameters:
    - size: `height`, `width` in pixels
    - position in the video: `left`, `top`, `right`, `bottom` in offset in pixels
2. Select the required image by ticking the radio button next to it.
3. Save the settings by clicking **OK** > **Save**.

4. Check that the player displays the logo.

IN THE FLUSSONIC CONFIGURATION FILE

You can also overlay a logo with the web player in the configuration file. To do that, follow these steps:

1. Upload the logo image using the `Flussonic-API: PUT /streamer/api/v3/logos/{name}` method.
2. Open the `flussonic.conf` configuration file.
3. In the stream settings, add the `logo` directive and specify the logo filename in the `path` parameter, starting with `@`.
4. Optional: Adjust the image size and its position in the video using these parameters:
    - size: `height`, `width` in pixels
    - position in the video: `left`, `top`, `right`, `bottom` in offset in pixels
5. Check that the player displays the logo.

```
stream example {
  input udp://239.0.0.1:1234;
  logo path=@logo.png height=100 width=100 left=0 top=0;
}
```

In this example:

- `path` is the logo file name, starting with `@`.
- (Optional) `height`, `width` specify the size of the logo image in pixels. If you specify one of these parameters, then the other scales proportionally. To display the logo in its original size, omit these parameters.
- (Optional) `left`, `top`, `right`, `bottom` specify the position of the logo image by offset in pixels. For example, to display the logo in the bottom right corner, specify the following values: `right=0, bottom=0`. Don't use the `left` and `right`, or the `top` and `bottom` parameters together.

Adding a logo to a video stream using the transcoder

When using the transcoder to overlay a logo on a video stream, the image is burned into the video track. It is displayed on any device and recorded in the DVR archive.

```
stream example {
  input udp://239.0.0.1:1234;
  transcoder vb=2048k logo=/storage/logo.png@10:10 ab=128k;
}
```

Here, `10:10` are the coordinates of the image, offset from the top left corner of the screen.

To place a logo in another part of the screen, you need to use a more complex notation. See the following examples:

- To place a logo in the center, use the following configuration:

```
stream example {
  input udp://239.0.0.1:1234;
  transcoder vb=2048k logo=/storage/logo.png@(main_w-overlay_w-10)/2:(main_h-overlay_h-10)/2 ab=128k;
}
```

- To place a logo in the bottom left corner, use the following configuration :

```
stream example {
  input udp://239.0.0.1:1234;
  transcoder vb=2048k logo=/storage/logo.png@10:(main_h-overlay_h-10) ab=128k;
}
```

- To place a logo in the top right corner, use the following configuration:

```
stream example {
  input udp://239.0.0.1:1234;
  transcoder vb=2048k logo=/storage/logo.png@(main_w-overlay_w-10):10 ab=128k;
}
```

- To place a logo in the bottom right corner, use the following configuration:

```
stream example {
  input udp://239.0.0.1:1234;
  transcoder vb=2048k logo=/storage/logo.png@(main_w-overlay_w-10):(main_h-overlay_h-10) ab=128k;
}
```

Learn more in the section Transcoder Settings.

3.8.13 Adding video to websites (embed.html)

Flussonic Media Server has a special page **embed.html**, which is intended for video insertion to a website and viewing video via a browser. Technically, `embed.html` is the same player that is used in the web interface of Flussonic Media Server.

The page with the player is available via the link:

`http://HOSTNAME/STREAMNAME/embed.html`

The page automatically detects a browser version and selects a supported video protocol. For the majority of devices it's HLS (the player uses it by default).

> **Warning**
>
> Video might start without sound due to the autoplay policy of browser vendors. The following link explains the policy and conditions for the sound to turn on automatically. See Chrome autoplay policy as an example

There are two ways to use the embed.html player:

- When opening **embed.html** directly (by entering the link in the address bar), the video will expand to the size of the browser window and start playing automatically.

- Also, you can use `embed.html` to embed the video on a website as a part of a web page. The HTML code for insertion is available on the **Overview** page of each stream in the web interface.

Example:

```
<iframe style="width:640px; height:480px;" allowfullscreen src="http://hostname/streamname/embed.html"></iframe>
```

The code embeds a player window with fixed dimensions (640x480px) to a web page. Playback starts automatically.

Options

For most tasks no additional configuration is required, but still `embed.html` has parameters that you can specify as part of the URL, for example:

```
http://FLUSSONIC-IP/STREAM_NAME/embed.html?autoplay=false&play_duration=600
```

In this example the video will be played without autostart and the playback will stop after 10 minutes.

The detailed description of all available parameters can be found in the Streaming API reference, in the **Query parameters** section.

If a stream has several audio, video, or subtitle tracks, you can use the `filter.tracks` parameter to specify which tracks to play.

Example of accessing the archive to view recorded video

To access a recording of a TV show, use the link with `from` and `to` parameters:

```
http://FLUSSONIC-IP/STREAM_NAME/embed.html?from=1511300552&to=1511300852
```

It is better to generate such links via server-side scripts, based on program guide (EPG), for the organization of a CatchUp service.

DVR player

To play a stream's DVR archive, open the player by using the link:

```
http://FLUSSONIC-IP/STREAM_NAME/embed.html?dvr=true
```

This DVR player plays recorded video from the archive, and it offers the calendar to navigate large archives in addition to the timeline.

The player interface allows you to set the timeline scale, enable fast playback, and save the specified interval as an MP4 file.

The DVR player supports all additional URL parameters, except **ago**.

The player interface allows you to automatically generate links in the format **embed.html?dvr=true&from=1511300552** without using additional tools. Just click a right time point on the timeline and click on the clock to open the link containing the **from** parameter.

See also:

- All ways to play an archive are described in the section DVR Playback

MULTIWINDOW MODE OF DVR PLAYER

> **Note**
>
> This functionality is available via the experimental `streams` parameter that probably will be changed soon. The actual list of `embed.html` parameters can be found in the Streaming API reference.

In some situations, it may be necessary to view DVR archives of several streams within one player with one timeline. For example, you may need to watch the recordings from several surveillance cameras synchronously to see the same location from different points of view. In this case you can use DVR player in multiwindow mode.

To do that, you can use the experimental `streams` parameter in the DVR player link and list all necessary streams in this parameter:

```
http://FLUSSONIC-IP/STREAM_NAME/embed.html?dvr=true&streams=cam01,cam02,cam03,cam04
```

> **Note**
>
> `STREAM_NAME` part in the URL may be replaced by the name of any stream, the displayed streams will be taken from the `streams` parameter.

This will result in playing all the archives in separate windows withing the DVR player.

Stream authorization

Flussonic Media Server has a built-in mechanisms for a basic protection against embedding video players on other websites. Please refer to Domain lock and CORS for player protection sections for detail on those.

3.8.14 Recommendations on developing the client application

You can use *Flussonic* as a signalling server when developing a website or an application for exchanging video and/or audio over WebRTC. For example, it may help you create an app for video conferencing ("many-to-many" connection), a website for conducting webinars or workshops ("one-to-many" connection), etc. Below you will find recommendations on using *Flussonic*'s WebRTC features in your project.

Make sure to configure *Flussonic* to accept publications from your app, and learn how you can play streams from *Flussonic* via WebRTC.

To write the code, use the **Flussonic WebRTC Player library**. It can be installed in one of the ways described below.

The description of the library classes and the example code can be found at npm.

Installing the library components via NPM and webpack

To import our library to your project with webpack, download the package:

```
npm install --save @flussonic/flussonic-webrtc-player
```

Then import components to your application:

```
import {
  PUBLISHER_EVENTS,
  PLAYER_EVENTS,
  Player,
  Publisher,
} from "@flussonic/flussonic-webrtc-player";
```

See also the demo application.

Installing the library components without NPM and webpack

Add this line to the script section of your HTML page:

`<script src="https://cdn.jsdelivr.net/npm/@flussonic/flussonic-webrtc-player/dist/index.min.js"></script>`

The example of a webpage containing the player code is below.

Player examples — with Webpack and without Webpack

Our demo application that uses Webpack to import components:

- **Sample app with Webpack and our WebRTC player**.

  In this example, the components are imported by Webpack into the application. You can download the application code and study how the player is implemented.

  Demo WebRTC player on JavaScript that obtains components via `<script>`:

- The *Flussonic* WebRTC player library code for implementing the WebRTC player is available in the CDN https://www.jsdelivr.com, and you can import it to your web page. To do this, add the following line to the script section of your HTML file `<script src="https://cdn.jsdelivr.net/npm/@flussonic/flussonic-webrtc-player/dist/index.min.js"></script>`.

The example of a page with the player in JavaScript (the similar code is included in the demo application):

```
<!DOCTYPE html>
<html>
  <head>


        <style>
      .app {
        display: flex;
        flex-direction: column;
        justify-content: space-between;
        height: calc(100vh - 16px);
      }
      .container {
        margin-bottom: 32px;
```

```
    }
    .video-container {
      display: flex;
    }
    .controls {
    }
    .config {
    }
    #player {
      width: 640px; height: 480px; border-radius: 1px
    }
    .button {
      height: 20px;
      width: 96px;
    }
    .preview {
      position: absolute;
      right: 0;
      bottom: 0;
      z-index: 100;
    }

    .preview-text {
      position: absolute;
      left: 0;
      top: 0;
      padding: 8px;
      background: black;
      color: white;
      z-index: 10;
    }

    #preview-video {
      width: 320px;
      height: auto;
      max-width: 320px;
      max-height: 240px;
    }
  </style>
  <script src="https://cdn.jsdelivr.net/npm/@flussonic/flussonic-webrtc-player/dist/index.js"></script>
</head>
<body>
  <div class="app">
  <div class="preview">
      <div class="preview-text">preview</div>
      <video id="preview-video" controls="false" muted autoplay playsinline />
  </div>
    <div class="video-container">
      <video
              id="player"
              controls
              muted
              autoplay
              playsinline
      >
      </video>
      <pre id="debug"></pre>
    </div>

  <div class="container">
    <div class="config" id="config">
      <span id="hostContainer">
        <label for="host">Host: </label><input name="host" id="host" value="" />
      </span>
      <span id="nameContainer">
        <label for="name">Stream: </label><input name="name" id="name" value="" />
      </span>
    </div>
    <div class="controls" id="controls">
      <select id="quality">
        <option value="4:3:240">4:3 320x240</option>
        <option value="4:3:360">4:3 480x360</option>
        <option value="4:3:480">4:3 640x480</option>
        <option value="16:9:360" selected>16:9 640x360</option>
        <option value="16:9:540">16:9 960x540</option>
        <option value="16:9:720">16:9 1280x720 HD</option>
      </select>
      <button id="publish" class="button">Publish</button>
      <button id="play" class="button">Play</button>
      <button id="stop" class="button">Stop all</button>
    </div>
    <div class="errorMessageContainer" id="errorMessageContainer"></div>
  </div>

<script>
  let wrtcPlayer = null;
  let publisher = null;

  const { Player, Publisher, PUBLISHER_EVENTS, PLAYER_EVENTS } = this.FlussonicWebRTC;

  const getHostElement = () => document.getElementById('host');
  const getHostContainerElement = () => document.getElementById('hostContainer');
  const getNameElement = () => document.getElementById('name');
  const getNameContainerElement = () => document.getElementById('nameContainer');
```

```
const getPlayerElement = () => document.getElementById('player');
const getPlayElement = () => document.getElementById('play');
const getPublishElement = () => document.getElementById('publish');
const getStopElement = () => document.getElementById('stop');
const getQualityElement = () => document.getElementById('stop');

const getStreamUrl = (
  hostElement = getHostElement(),
  nameElement = getNameElement(),
) =>
  `${hostElement && hostElement.value}/${nameElement && nameElement.value}`;
const getPublisherOpts = () => {
  const [, , height] = document.getElementById('quality').value.split(/:/);
  return {
    preview: document.getElementById('preview-video'),
    constraints: {
      // video: {
      //   height: { exact: height }
      // },
      video: true,
      audio: true,
    },
    canvasCallback: (canvasElement) => {
        window.myCanvasElement = canvasElement;
    },
  };
};

const getPlayer = (
  playerElement = getPlayerElement(),
  streamUrl = getStreamUrl(),
  playerOpts = {
    retryMax: 10,
    retryDelay: 1000,
  },
  shouldLog = true,
  log = (...defaultMessages) => (...passedMessages) =>
    console.log(...[...defaultMessages, ...passedMessages]),
) => {
  const player = new Player(playerElement, streamUrl, playerOpts, true);
  player.on(PLAYER_EVENTS.PLAY, log('Started playing', streamUrl));
  player.on(PLAYER_EVENTS.DEBUG, log('Debugging play'));
  return player;
};

const stopPublishing = () => {
  if (publisher) {
    publisher.stop && publisher.stop();
    publisher = null;
  }
};

const stopPlaying = () => {
  if (wrtcPlayer) {
    wrtcPlayer.destroy && wrtcPlayer.destroy();
    wrtcPlayer = null;
  }
};

const stop = () => {
  stopPublishing();
  stopPlaying();

  getPublishElement().innerText = 'Publish';
  getPlayElement().innerText = 'Play';
};

const play = () => {
  wrtcPlayer = getPlayer();
  getPlayElement().innerText = 'Playing...';
  wrtcPlayer.play();
};

const publish = () => {
  if (publisher) publisher.stop();

  publisher = new Publisher(getStreamUrl(), getPublisherOpts(), true);
  publisher.on(PUBLISHER_EVENTS.STREAMING, streaming);
  publisher.start();
};

const setDefaultValues = () => {
    getHostElement().value = config.host;
    getNameElement().value = config.name;
};

const setEventListeners = () => {
  // Set event listeners
  getPublishElement().addEventListener('click', publish);
  getPlayElement().addEventListener('click', play);
  getStopElement().addEventListener('click', stop);
  getQualityElement().onchange = publish;
};
```

```
    const main = () => {
      setDefaultValues();
      setEventListeners();
    };

    const streaming = () => {
        getPublishElement().innerText = 'Publishing...';
        getPublishElement().disabled = true;

        // drawing on publishing canvas
        if (window.myCanvasElement) {
          const ctx = window.myCanvasElement.getContext('2d', { alpha: false });
          ctx.filter = 'sepia(0.75)'; // Testing filters
          ctx.font = '128px sans-serif';
          ctx.fillStyle = 'white';
          ctx.textAlign = 'center';
          ctx.textBaseline = 'center';
          (function loop() {
            ctx.fillText(
              `It's publishing to Flussonic!`,
              window.myCanvasElement.width / 2,
              window.myCanvasElement.height / 2,
              window.myCanvasElement.width - 100,
            );
            setTimeout(requestAnimationFrame(loop), 1000 / 30); // drawing at 30fps
          })();
      }
    };

    window.addEventListener('load', main);
  </script>
    </body>
</html>
```

Copy this code to a file, for example `index.html`, and open in the browser to check how the player works.

3.8.15 MSE player

Here we will tell how to use our open-sourced MSE player in your applications to provide low latency video playback in the browser. The player has long been offered in our `embed.html` mechanism.

Why MSE Player?

1. Uses HTML5 and doesn't require Flash, which means it is supported by any client device (browser, smartphone)
2. Has a list of advantages in comparison with WebRTC, such as: WebRTC requires UDP, and MSE requires HTTP, which makes it easier for MSE to pass through corporate firewalls and proxies. In addition, WebRTC and MSE support different codecs, for example, audio: MSE supports AAC audio codec, as opposed to WebRTC. It means that a playback of a TV channel is easier with the help of the MSE Player.

You can see the MSE Player in some parts of *Flussonic* and *Watcher UI* and can also access it from the browser via the following URL:

```
http://flussonic-ip/STREAMNAME/embed.html?realtime=true
```

The mechanism that is used by *Flussonic* is described in HTML5 (MSE-LD) Low Latency Playback

You can use its JavaScript module in your frontend projects. The sources are published to Github.

On this page:

Installation in your app

Step 1.
Run the following command:

```
npm install --save @flussonic/flussonic-mse-player
```

Step 2.
Import it into JS:

```
import FlussonicMsePlayer from '@flussonic/flussonic-mse-player'
...
const player = new FlussonicMsePlayer(element, url, opts)
```

Sample app with webpack and our MSE player

You can find the source code of MSE Player on Github.

The FlussonicMsePlayer class

```
var player = new FlussonicMsePlayer(element, streamUrl, opts)
```

Parameters:

- `element` — `<video>` a DOM element
- `streamUrl` — the URL of a stream
- `opts` — player options.

You can monitor MSE Player with Sentry, setting the `sentryConfig(string)` parameter (see the Table below).
Player options (`opts`) include the following settings:

Parameters	Type	Description
`progressUpdateTime`	integer (seconds)	time period after which the player will provide the information about the playback progress
`errorsBeforeStop`	integer	number of playback errors that will be processed by the player until a complete stop
`connectionRetries`	integer	number of retries to establish a connection before the player stops
`preferHQ`	boolean	if set to `true`, player will automatically select the highest available quality of the stream
`retryMuted`	boolean	if set to `true`, player will try to restart the playing process with initialy muted sound
`maxBufferDelay`	integer	maximum buffer delay value. If a live playback lags behind the real time by more than the specified value, the excess is discarded
`sentryConfig`	string	DSN from Sentry

METHODS

Method	Description
`play()`	start playing
`stop()`	stop playing
`setTracks([videoTrackId, audioTrackId])`	set up video and audio tracks for a playback
`getVideoTracks()`	return available video tracks (should be used in the `onMediaInfo` callback method)
`getAudioTracks()`	return available audio tracks (should be used in the `onMediaInfo` callback method)

EVENT CALLBACKS

Event	Description
`onProgress(currentTime)`	triggered every 100ms while a stream is playing and gives current playback time
`onMediaInfo(metadata)`	triggered when `metadata` of the stream is available. The `metadata` includes a common information of the stream such as width, height, information about mbr streams and so on. After this callback triggered you can use `getVideoTracks()/getAudioTracks()` methods

Using mutli-bitrate tracks

Let's consider a video stream that has three video tracks: v1(800k), v2(400k), v3(200k) and two audio tracks: a1(32k), a2(16k).

To set default tracks to v2 and a1, add the `tracks` URL parameter with track numbers:

`'ws://flussonic-ip/stream_name/mse_ld?tracks=v2a1'`

And then pass this URL to the player constructor.

You can get all available video/audio tracks:

- inside `onMediaInfo(metadata)` callback, by parsing `metadata`:

```
{
  width: ...,
  height: ...,
  streams: [
    {
      track_id: "v1", bitrate: ..., codec: ..., content: "video", fps: ..., ...
      ...

      track_id: "a1", bitrate: ..., codec: ..., content: "audio", fps: ..., ...
    }
```

```
  ]
}
```

- inside `onMediaInfo(metadata)` by calling `getVideoTracks()/getAudioTracks()` methods.

To set tracks for a playback, use the `setTracks([videoTrackId, audioTrackId])` method.

Complete example

```
<html>
  <head>

  </head>
  <style>
    .player-container {
      border: 1px solid black;
    }

    #player {
      position: relative;
      width: 100%;
    }

    .mbr-controls {
      display: none;
    }
  </style>
<body>
  <div class="player-container">
    <video id="player"/>
  </div>
  <div class="mbr-controls">
    <div>
      <label for="videoTracks">video tracks</label>
      <select name="videoTracks" id="videoTracks"></select>
    </div>
    <div>
      <label for="audioTracks">audio tracks</label>
      <select name="audioTracks" id="audioTracks"></select>
    </div>
    <button onclick="window.setTracks()">set tracks</button>
  </div>

  <button onclick="window.player.play()">Play</button>
  <button onclick="window.player.stop()">Stop</button>
  <script type="text/javascript" src="/flu/assets/FlussonicMsePlayer.js"></script>
  <script>
    window.onload = onLoad();

      function onLoad() {

        var element = document.getElementById('player');
        var videoTracksSelect = document.getElementById('videoTracks');
        var audioTracksSelect = document.getElementById('audioTracks');
        var mbrControls = document.querySelector('.mbr-controls');

        var url = (window.location.protocol == "https:" ? "wss:" : "ws:")+ '//'+window.location.host+'/clock/mse_ld';

        window.player = new FlussonicMsePlayer(element, url);

        window.player.onProgress = function(currentTime) {
          console.log(currentTime);
        };

        window.player.onMediaInfo = (rawMetaData) => {
          var videoTracks = window.player.getVideoTracks()
          var audioTracks = window.player.getAudioTracks()
          var videoOptions = videoTracks.map((v, i) => (
            `<option value="${v['track_id']}">${v['bitrate']} ${v['codec']} ${v['fps']} ${v['width']}x${v['height']}</option>`
          ))

          var audioOptions = audioTracks.map(v => (
            `<option value="${v['track_id']}">${v['bitrate']} ${v['codec']} ${v['lang']}</option>`
          ))

          videoTracksSelect.innerHTML = videoOptions.join('')
          audioTracksSelect.innerHTML = audioOptions.join('')

          mbrControls.style.display = 'block'
        }

        window.setTracks = () => {
          var videoTrackId = videoTracksSelect.options[videoTracksSelect.selectedIndex].value
          var audioTrackId = audioTracksSelect.options[audioTracksSelect.selectedIndex].value

          window.player.setTracks([videoTrackId, audioTrackId])
        }
      }
  </script>
</body>
```

```
</html>
```

Statistics on the MSE Player

When you initialize the player, add the `config` variable:

```
this.player = new FlussonicMsePlayer(this._videoElement, url, config);
```

The MSE Player has the `onStats` option that should be passed to the `config` parameter. It returns an object, containing statistics on the player's buffers and the timestamp of when the statistics was obtained.

Adding controls like in a desktop player (Flussonic 20.10)

The MSE Player now supports new controls the same as found in usual desktop players, such as pause, resume or unmute. The controls are part of *MediaElement*, which can be attached to the player as a separate part after initializing.

Using the `attachMedia(element)` method, you can attach a `<video />` element to the player separately, after initializing the player. You can also pass the *MediaElement* through the player parameters, then it will be attached automatically.

To control the player via *MediaElement*, you will need the following events: `onMediaAttached`, `onPause`, `onResume`.

Using the `onMediaAttached` event, you know exactly when the player is attached to the `<video />` and is ready to start playing.

Here is a usage example:

```
onMediaAttached: () => {
      element.play()
    },
```

The player listens to the native HTTP events of the `<video />` element (in which you pass the player to the web page), such as `play/pause/unmute`, and can respond to them. Using `onPause` and `onResume`, you can respond to pause and resume playback events. For example, you can draw interface elements (like a large pause icon).

3.8.16 Low-latency broadcasting to a large audience

LL-HLS is suitable for low-latency broadcasting to a large audience. Typical scenarios:

- Broadcasting a sports match: thousands of viewers, they are not ready to hear "goal!" later than their neighbors.
- Online conferences, large webinars: many viewers interacting through chat, and latency can hinder communication with the audience.

For such tasks, we offer LL-HLS: it allows you to build an efficient CDN and keep acceptable latency.

Stream preparation

Before playing, configure the publication in Flussonic. Most often, publication is done:

- From programs like OBS. Here is an article on how to do this.
- From a browser, with a webcam or screen presentation. On the Publishing via WebRTC page, we explained how to set this up.

Server preparation

LL-HLS requires HTTP/2, which means you need to configure HTTPS on the server and provide the player with an `https://` link. Some players ignore this requirement, but we recommend not skipping this:

- Safari will definitely not work without https. It will switch to regular HLS mode, and low latency will be lost.
- HTTP/2 allows multiplexing requests in one connection. This provides a better user experience: the video will buffer less frequently.

Here is an article that will help you quickly set up https.

Playing with low latency via LL-HLS

The LL-HLS stream can be played:

- In any third-party player with LL-HLS support. The LL-HLS playback link can be copied from the **Output** tab in the stream profile.
- In our `embed.html` player, if you add the parameters `realtime=true&proto=ll-hls` to the URL. Open this link in a browser on a phone or computer:

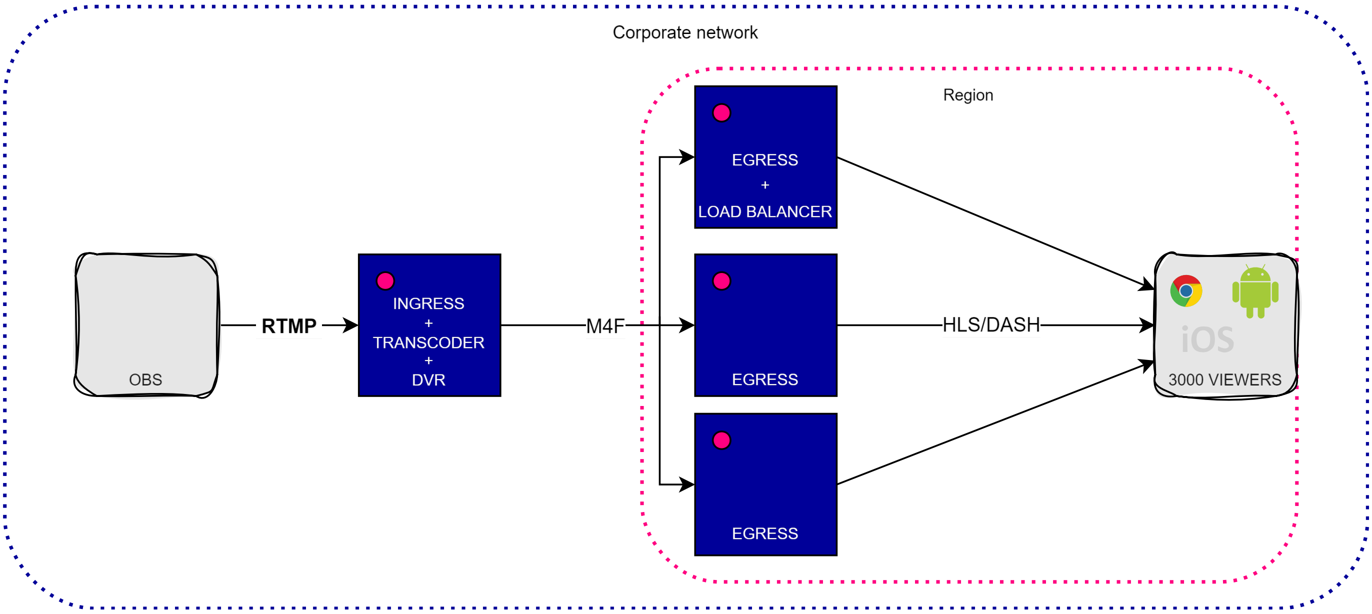
```
https://FLUSSONIC-IP/STREAMNAME/embed.html?realtime=true&proto=ll-hls
```

3.8.17 Streaming to thousands of employees

When you're conducting a private broadcast just for company employees or partners in various cities and countries:

- **Use your own or rented servers** to prevent the video from being accessible on external platforms like YouTube or Vimeo.
- **Set up one or more edge servers** in each branch depending on the number of viewers to avoid the need for a large Internet bandwidth on origin server.
- **Adapt the video for various devices** to ensure smooth playback even on small screens or with poor connections.
- **Enable recording for rewinding** so that latecomers can watch the broadcast from the beginning.

This page will explain how to achieve all this using Flussonic Media Server.



Ingest the video

- **[Install Flussonic Media Server](#)** on your server.
- **Configure** [publishing](#).
- **Publish the video to Flussonic** from software like OBS Studio or vMix via any protocol: RTMP, WebRTC WHIP, SRT, etc.

On the same server (origin) where the ingest is performed, configure transcoding and recording. Using a single server for these functions is possible because you are publishing only one or a few streams for corporate streaming; so ingesting, transcoding and recording will not require many resources.

Adapting the video to various devices

Video adaptation involves transcoding to multibitrate, which allows the stream to adjust to the viewer's resolution and internet connection, fro example play in lower quality if the connection deteriorates. Use Flussonic Media Server to transcode the video and obtain a multibitrate stream.

- **Transcode on GPU NVENC or Intel Quick Sync Video**, since hardware support guarantees optimal use of server resources.
- **Rent a server with GPU or QSV** in the cloud or use your own server.

Record your broadcast

Recording will allow viewers to use rewind. Recording may also be a regulatory requirement, for example if you are recording a shareholders meeting.

- **Configure recording** on the origin server. A transcoded multibitrate stream shall be recorded in the archive, so that all bitrates were available when rewinding and timeshifting.

Rent servers for restreaming

One origin server will not be able to distribute video to all tens of thousands of viewers. Use your own servers in your branches or rent edge servers in broadcasting regions:

- **Plan your capacity**. Count on approximately 3-5 servers per 5000 viewers. To understand exactly how many and what kind of edge servers you need, contact our sales and technical support department at support@flussonic.com.
- **Install Flussonic Media Server** on the edge servers.
- **Configure restreaming** from origin to edge.
- **Configure caching**. Edge servers should have an SSD for caching the archive to reduce the load on the DVR on the origin server.

Set up balancing

To provide a single URL for playback in all regions, while preventing any one server from being overloaded by directing users to the closest available server:

- **Configure load balancing** to distribute playback requests between several servers in one region.
- **Configure geo balancing** for the edge servers to handle playback requests from their region only.

Provide viewers with the URL

Videos from Flussonic can be played on smartphones and computers, and even fed to the branch cable network.

- **Use our embed.html player** on your website to play in a browser on smartphones and computers.
- **For other players, use links for playback** over various protocols, including HLS/DASH, or low latency protocols MSE-LD, WebRTC, LL-HLS.

3.9 Protection

3.9.1 Authorization

Authorization is a security mechanism, enabling you to grant permissions to a user to do or access something.

Flussonic Media Server identifies users and tracks connections using ***authorization backends***. Authorization backend establishes the authorization rules to allow/deny the requests. Authorization backend may be an external system (for example, your site), a custom script on the disk, a part of Flussonic configuration, or IPTV plugin — in any case, it defines the authorization rules.

The main idea of authorization in Flussonic is as follow: you site recognizes a client that is going to play video (for example, by cookies) and adds a unique token to Flussonic URL used in the player. Then, the player requests the video from Flussonic with this token. Flussonic sends a request to an authorization backend (for example, back to your site) to find out if the player is allowed to play the video with this token. If the authorization backend allows playback, the client gets access to the video.

To see how *Flussonic Media Server* interacts with authorization backend, watch the video below:

type:video

For more details about interaction of Flussonic Media Server with an authorization backend, please refer to the Authorization using a backend section.

Flussonic uses HTTP features for HLS protocol and handles persistent TCP sessions for RTMP, RTSP, and MPEG-TS. *Flussonic* also tracks export of MPEG-TS and MP4 segments from a DVR archive.

In addition, *Flussonic Media Server* has a built-in mechanisms for a basic protection against embedding video players on other websites. Please refer to Domain lock and CORS for player protection sections for detail on those.

Flussonic Media Server can also check for a password when publishing a stream. Find more details about this feature at Protecting a publication with a password.

Authorization using a backend

Authorization backend establishes the authorization rules to allow/deny the requests.

Flussonic allows to configure either *external* or *internal authorization*:

External authorization is used if you prefer an **external system** to approve or reject requests. In this case *Flussonic* does **not know** who is allowed to access a stream. The algorithm is like so:

1) User requests access to a stream from *Flussonic*. 2) *Flussonic* reaches out to authorization backend. 3) The backend checks if the user is allowed to access the stream and returns the corresponding response. 4) *Flussonic* either grants or restricts access for the user.

Internal authorization is used if you prefer *Flussonic Media Server* to approve or reject requests. Now *Flussonic* **knows** who is allowed to access a stream. For more information, see Authorization Configurator.

Flussonic Media Server supports two authorization backends:

- Playback session authorization (`on_play`)

Limits the access to the playback for unauthorized users. See: Playback authorization for more details.

- Publish session authorization (`on_publish`)

Limits the access to publishing for unauthorized users. See: Publish authorization for more details.

DESCRIPTION OF THE AUTHORIZATION PROCEDURE

Step 1.

Add the Flash Player or HTML `<video>` tag on your website or Middleware, and specify the path to a video with an authorization key (`token`), created on this website, according to one of the following formats:

- query string for HLS, HTTP MPEG-TS, and other HTTP-based protocols:

`http://192.168.2.3:80/stream1/index.m3u8?token=60334b207baa` for HLS

- RTMP address: `rtmp application rtmp://192.168.2.3/static stream name: stream1?token=60334b207baa`
- RTSP address: `rtsp://192.168.2.3/stream1?token=60334b207baa`

If your website or Middleware does **not** use tokens in a video path, *Flussonic Media Server* will create one automatically.

Step 2.

After receiving a request with a token, *Flussonic Media Server* checks whether a session is already open (stream is already broadcasted from the server to the client) by the user. It does so with the help of the *session identifier*. ***Session identifier*** is a unique ID that server assigns to the session to identify and track user's activity.

A ***session identifier*** is a hash sum created with stream name (`name`), client's IP address (`ip`) and token (`token`) as follows:

hash(name + ip + token)

Therefore, if a user changes the IP address or switches to another stream, a new session is created.

Step 3.

If there are no open sessions, *Flussonic Media Server* sends a request to auth backend. The whole list of the parameters sent in the request can be found in Authorization Backend API schema.

Step 4.

Backend returns a response that contains the following:

1. Information about the session: during which period the session is alive, how many opened sessions are allowed with this user id.
2. Configuration of injecting advertising video clips into played stream. See Advertisement Insertion.

The whole list of the parameters received in the response can be found in Authorization Backend API schema.

HOW TO ENABLE AUTH BACKEND

To enable the backend add the `on_play` directive to the configuration file:

```
on_play PATH_TO_AUTH;
```

where **PATH_TO_AUTH** may take the following values:

- **HTTP address**

Flussonic Media Server makes HTTP requests to this address and passes session parameters to the backend.

HOW TO LEAVE THE SAME SESSION ON MOBILE ROAMING WITH CHANGING IP

The generation of session identifiers (IDs) is dependent on tokens. So if the token changes, the session ID changes consequently. But what if we need to skip reauthorization and session change on changing client IP address on mobile roaming? It is possible with the help of session keys that are used to generate the session ID.

You may use the following session keys:

Key	Description
`proto`	protocol
`name`	stream name
`ip`	IP-address
`token`	token

The session ID is a hash sum with the following formulae:

hash(name + ip + proto + token)

Thus, if any of the keys (not necessarily the token) changes, the session will be finished, and a new session will be opened.

To specify session keys, go **Media** > click the stream name > **Auth** > select session keys from the **Select session keys** drop-down list.

You can also specify the `session_keys` parameter in the `on_play` URL settings, in the stream configuration.

The following restrictions apply to the list of elements (keys) for `session_keys`:

- `name`, and `proto` are **required** for `session_keys` and must be specified **explicitly** (order does not matter);
- duplicate values are **allowed** and processed explicitly, which affects the final result;
- the list items are separated by comma (`,`), without spaces;
- if a key value does **not** exist, the `undefined` value is used.

Let's have a look at the ***example***:

```
stream example_stream {
  input fake://fake;
  on_play http://IP-ADDRESS:PORT/php-auth-script.php session_keys=ip,name,proto,token;
}
```

In the example above `ip`, `name`, `proto`, and `token` are used to generate the session ID (`session_id`).

SESSION IS OPENED

If the backend allows opening of the session, by default *Flussonic Media Server* will re-check session **every 3 minutes** to determine the session is still active.

You can send an `X-AuthDuration` HTTP header to change this time. `X-AuthDuration` is specified in seconds.

Upon 3 minutes (or another period, if it has been modified with `X-AuthDuration`) request is repeated. If the backend is not available or returns the `HTTP 500` response, *Flussonic Media Server* keeps the previously received status from the backend and sends the request again.

SESSION IS CLOSED

If the backend banned the session, the information about this session is cached on the server.
If the user tries to open stream again with the same token, *Flussonic Media Server* will reject it without making any new calls to the backend.

WEB INTERFACE NOTES

The Administrator can watch any video in the *Flussonic* web interface without authorization. That is, the authorization backend is **not** used in this case.

Technically, this is implemented as follows: when the Admin accesses video in the web interface, a special token `ADM-xxx` is generated, which is intercepted by *Flussonic Media Server*.
Such a token is perceived as permission to play video without authorization.

You can prevent the Administrator from viewing videos protected by the backend authorization mechanism.

Example of auth script (PHP)

Let's store credentials in the `auth.txt` file, pre-populated with the following data:

```
user1:token1
user2:token2
user3:token3
```

The following PHP script checks whether a token is in this file, and allow the opening of a session for existing tokens:

```
<?php

$get = print_r($_GET, true);
$token = $_GET["token"];
if(!$token || !strlen($token)) {
    header('HTTP/1.0 403 Forbidden');
    error_log("No token provided", 4);
    die();
}

$tokens = array();
$contents = explode("\n", file_get_contents("auth.txt"));
foreach($contents as $line) {
    if(strlen($line) > 3) {
        $parts = explode(":", $line);
        $tokens[$parts[1]] = $parts[0];
    }
}

if($tokens[$token]) {
    header("HTTP/1.0 200 OK");
    header("X-UserId: ".$tokens[$token]."\r\n");
    header("X-Max-Sessions: 1\r\n"); // Turn this on to protect from multiscreen
} else {
    header('HTTP/1.0 403 Forbidden');
}
?>
```

Gathering statistics using X-UserId

When a new session is opened, the backend may send the `X-UserId` HTTP header to a *Flussonic Media Server* (e. g., `X-UserId: 100`), that will be stored in the internal database with the session data when this session is closed. To gather statistics, you can request information about a session using MySQL protocol and `X-UserId`.

If a backend sends `X-Unique: true` alongside `X-UserId`, it will close all the other open sessions, having the same `X-UserId`.

> **Warning**
>
> Disconnected sessions remain in a memory of a server for some time. Therefore, clients with the same combinations of IP address, stream name, and token are **not** able to access the content.

If you use `X-Unique` you should generate different tokens each time a user accesses a page.

What happened on auth backend timeout?

When the authorization backend fails to reply within 3 seconds, the following situations may occur:

Session state	Result
not opened yet	Is not open or forbidden
allowed	Remains allowed
forbidden	Remains forbidden

Enabling playback and publish sessions authorization

Flussonic supports playback and publishing sessions authorization.

PLAYBACK SESSION AUTHORIZATION

Before playing the stream, *Flussonic* has to ensure a user has access rights to watch the content.

When a client requests a stream for playback, *Flussonic* sends a request to the authorization backend. If the stream is up and running and a client has permission to watch it, the authorization backend returns HTTP 200, and *Flussonic* sends a stream URL to the client.

To enable the playback authorization, use the `on_play` option in the `template` or `stream` settings like so:

```
template on-play-example {
  on_play http://IP-ADDRESS:PORT/PATH_TO_SCRIPT;
}
```

PUBLISHING SESSION AUTHORIZATION

Flussonic allows you to authorize publishers via third-party software to avoid unreliable sources and give the permissions only to trustworthy publishers. Only authorized users can publish the content.

It works as follows: a user establishes connection with *Flussonic Media Server* and requests permission to start publishing the stream. *Flussonic* sends a `POST` request to an HTTP handler. JSON body of the request contains fields listed at API Schema. It is basically a set of parameters identifying the session to check if a user is allowed to publish the stream. Based on the response from the authorization backend (`HTTP 200` or `HTTP 403`), *Flussonic* either allows or forbids the publishing for the user.

To configure the publishing session authorization, use the `on_publish` option in the `template` or `stream` settings as follows:

```
template on-publish-example {
  prefix on-publish-example;
  input publish://;
  on_publish http://IP-ADDRESS:PORT/on_publish.json;
}
```

3.9.2 Authorization configurator

You can declare complicated authorization settings right in the *Flussonic* configuration file.

You can specify black and white lists of IP addresses, tokens, User-Agents, and countries, and include multiple parallel authorization HTTP backends. You don't need to write your own scripts.

Setting up authorization

Add these lines to `/etc/flussonic/flussonic.conf`:

```
auth_backend myauth1 {
  allow ip 127.0.0.1;
  allow ip 192.168.0.1;
  allow ip 172.16/24;
  deny ip 8.8.8.8;
  allow country RU US;
  deny country GB;
  allow token test_token1;
  deny ua "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.10)";
  backend http://stalker-1.iptv.net/auth.php;
  backend http://stalker-2.iptv.net/auth.php;
}
```

- `allow` – declares the white list.
- `deny` – declares the black list.

Flussonic applies the rules in the following order:

- allow token
- deny token
- allow ip
- deny ip
- allow country
- deny country
- allow useragent
- deny useragent
- Makes requests to parallel backends
- If `allow default` was not specified, then denies access.

The rule priority matters. Rules with a higher priority are applied immediately, and then rules with a lower priority are no longer taken into account. For example, if you allow the client's IP address but the client's token is in the black list — the access will be denied because the token has a higher priority.

By the *client* we mean a client application or device that receives video from the *Flussonic* server.

To apply this auth backend to a stream, specify `auth://myauth1`:

```
stream example_stream {
  input udp://239.255.0.1:1234;
  on_play auth://myauth1;
}
```

Rules will be applied after you reload the configuration.

The 'allow default' option

The `allow default` option defines the default behavior in the case when all backends are not responding (for example, because of an error in an HTTP response or non-working script). If this option is enabled, all clients or devices except those listed explicitly in the `deny` option will have access to the content. And if this option is disabled, all clients or devices except those listed explicitly in the `allow` option will not have access to the content.

In this way, the `allow default` option gives you the opportunity to access the content in case the backend is not working.

Let's see how *Flussonic* deals with different responses from the backend and how the enabled `allow default` option affects the decision to grant access to a video stream.

ALLOW DEFAULT OPTION IN CASE OF ONE BACKEND

If the authorization backend denies access (responds with an error code 4xx, such as `403 Forbidden`), *Flussonic* doesn't allow access to the content, even if you have enabled `allow default` in the stream settings.

However, if the backend is down (does not respond due to an error) or there is a server error on the server where the backend script runs (with an error code 5xx, such as `500 Internal Server Error`), *Flussonic* allows access to the content to all clients (recipients) except those listed in the `deny` option.

ALLOW DEFAULT OPTION IN CASE OF MULTIPLE BACKENDS

If there are multiple parallel backends, the rules are similar.

If at least one of the backends allows access, access will be granted, even if other backends deny it or are not responding.

If at least one of the backends denies access, and all other backends are not responding (no one allows it), access will be denied.

However, if *all* backends are down (not responding), *Flussonic* allows access to the content to all clients except those listed in the `deny` option.

This table illustrates the logic of authorization in case of using multiple authorization backends on a stream:

Backend 1	Backend 2	Backend 3	Resulting answer
allow	allow	allow	Allow
ban	ban	ban	Ban
ban	allow	ban	Allow
not responding	not responding	not responding	Allow
not responding	allow	not responding	Allow
not responding	ban	not responding	Ban

Examples

MULTIAUTH HTTP AND ACCESS FROM A LOCAL NETWORK

```
auth_backend multi_local {
  allow ip 192.168.0/24;
  backend iptv://localhost; # iptv plugin
  backend http://examplehost/stalker_portal/server/api/chk_flussonic_tmp_link.php;
}
```

BAN SOME IP ADDRESSES

```
auth_backend blacklist {
  deny ip 1.1.1.1;
  deny ip 2.2.2.2;
  deny ip 10.10/16;
  allow default;
}
```

USE AN HTTP BACKEND AND ALLOW VIDEO TO CLIENTS WITH THE SPECIFIED TOKENS

```
auth_backend myauth2 {
  allow token friend_token1;
  allow token friend_token2;
  backend http://examplehost/stalker_portal/server/api/chk_flussonic_tmp_link.php;
}
```

ALLOW SOME USER-AGENTS (CERTAIN SET-TOP-BOXES) AND BLOCK OTHERS

```
auth_backend agents {
  allow ua MAG;
```

```
  allow ua TVIP;
}
```

3.9.3 Stalker Middleware and Flussonic

Stalker Middleware

Stalker — popular free IPTV Middleware from Infomir company. Stalker works with our DVR and auth system.

This article will help you configure Stalker with Flussonic.

Authorization

ON FLUSSONIC

On Flussonic side just add one line to flussonic.conf:

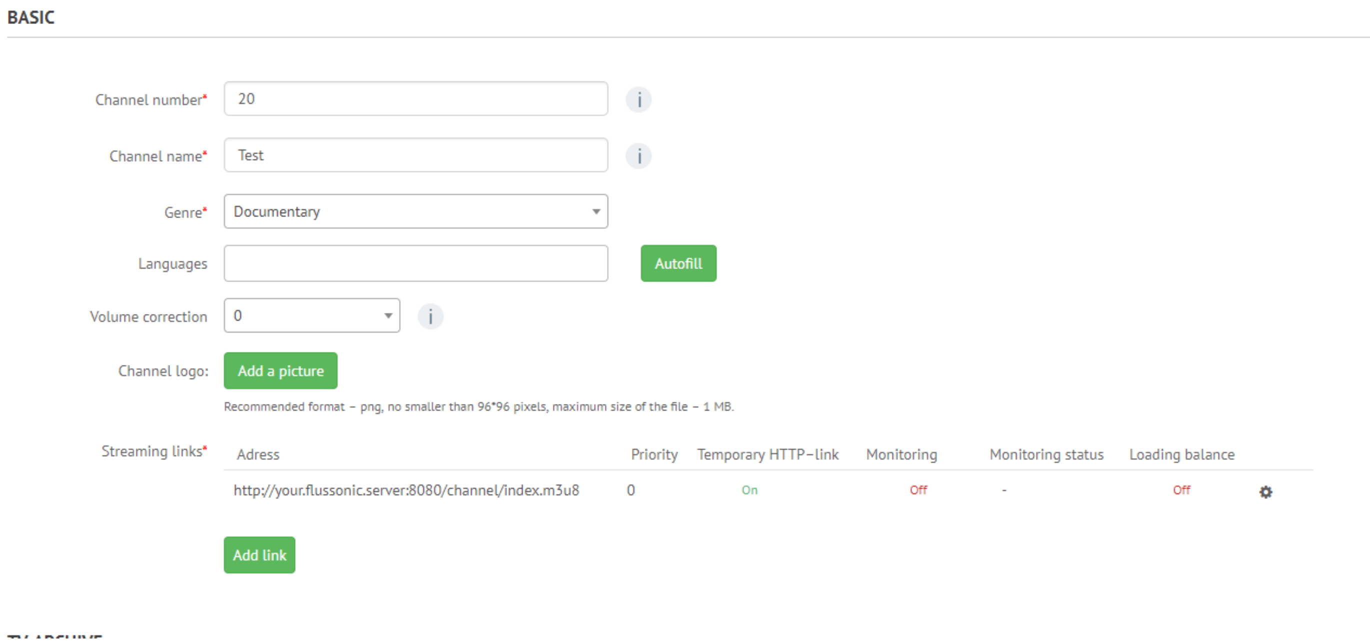
```
on_play http://<stalker_host>/stalker_portal/server/api/chk_flussonic_tmp_link.php;
```

and then reload the configuration:

```
service flussonic reload
```

ON STALKER

When you create/edit channel in **Streaming links** you need to set the option **Flussonic** in **Temporary URL**.



This setting is finished. Now in Flussonic admin panel you can see what users are using tokens for authentication.

DVR

ON FLUSSONIC

Additional configuration is not required. Just make sure you have enabled DVR on necessary channels.

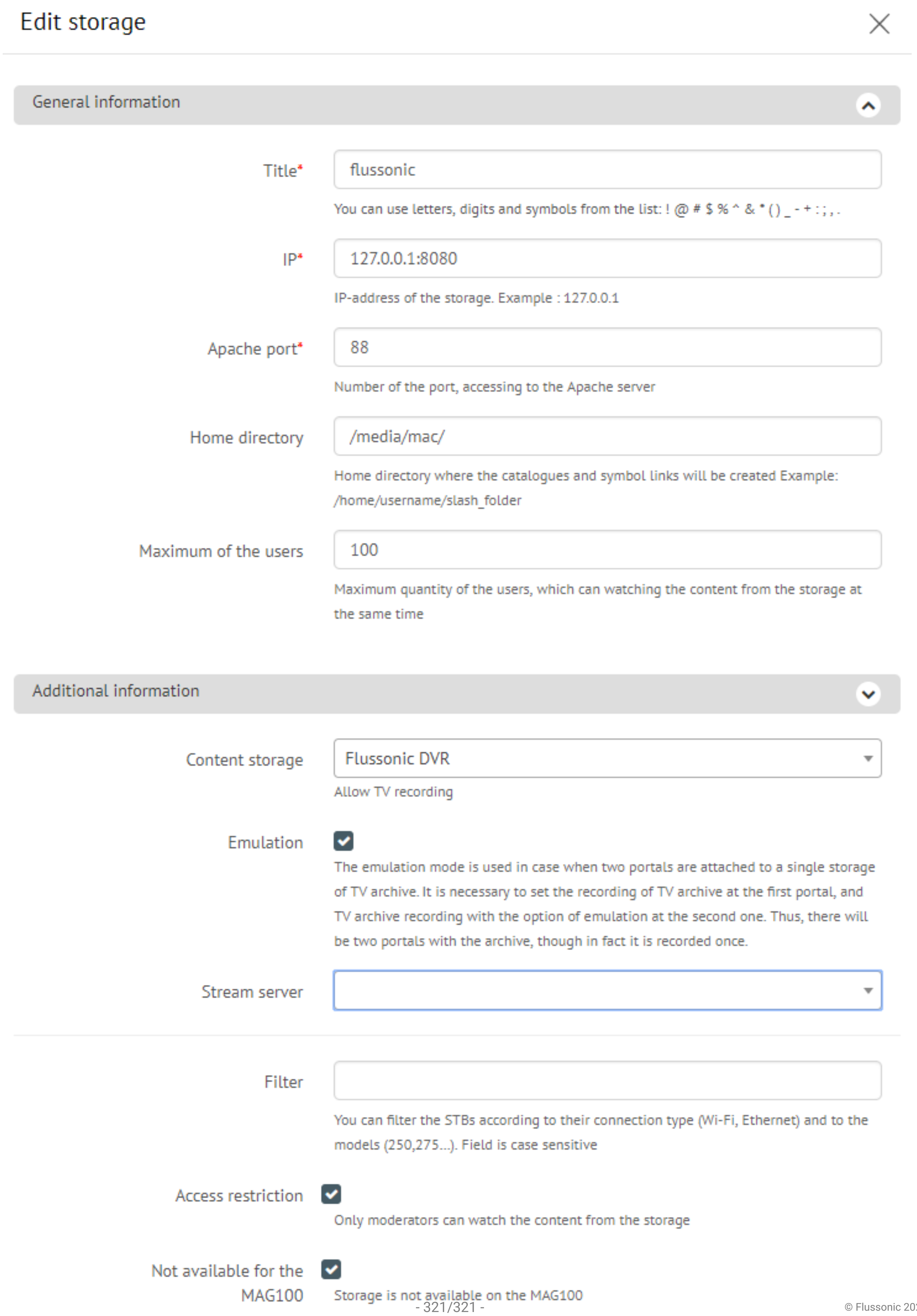
ON STALKER

- Add Storage:

In Stalker admin panel go to the menu **Storage** > **Storage list**.

Click the **Add storage** button.

Fill the required fields **Title**, **IP**, **Port** and in the **Additional information** tab select **Flussonic DVR** from **Content storage**.


Edit storage
General information
Title*
flussonic
You can use letters, digits and symbols from the list: ! @ # $ % ^ & * ( ) _ - + : ; , .
IP*
127.0.0.1:8080
IP-address of the storage. Example : 127.0.0.1
Apache port*
88
Number of the port, accessing to the Apache server
Home directory
/media/mac/
Home directory where the catalogues and symbol links will be created Example: /home/username/slash_folder
Maximum of the users
100
Maximum quantity of the users, which can watching the content from the storage at the same time
Additional information
Content storage
Flussonic DVR
Allow TV recording
Emulation
The emulation mode is used in case when two portals are attached to a single storage of TV archive. It is necessary to set the recording of TV archive at the first portal, and TV archive recording with the option of emulation at the second one. Thus, there will be two portals with the archive, though in fact it is recorded once.
Stream server
Filter
You can filter the STBs according to their connection type (Wi-Fi, Ethernet) and to the models (250,275...). Field is case sensitive
Access restriction
Only moderators can watch the content from the storage
Not available for the MAG100
Storage is not available on the MAG100