
AURA - Automated Radio

The AURA Team

Apr 19, 2024

GUIDES

1	User Guide	3
2	Administration Guide	27
3	Developer Guide	57
4	Release Notes	77
5	Bug Reports	89
6	Contributing to AURA	91
7	Contributor Covenant Code of Conduct	93
8	About AURA	97
9	Matrix	99
10	Mailinglist	101
11	Partners	103

AURA is a software suite for community radio stations. All code is Open Source and licensed under [AGPL 3.0](#).

Alpha Status

We are currently in alpha status, so do not use AURA in production yet. If you are an early-stage user or developer, we would be happy about your *contributions*.

Get an overview on the current release at the [Release Notes](#) page.

USER GUIDE

This guide covers the user interface for radio hosts and programme managers.

1.1 Dashboard overview

The dashboard is a browser application allowing:

- *Hosts* to upload and manage their shows.
- *Programme Managers* to manage the radio station and its programme.

1.1.1 Requirements

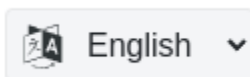
Dashboard is currently supported by these browsers:

- Chrome 115+
- Firefox 122+

At the moment mobile browsers are not supported.

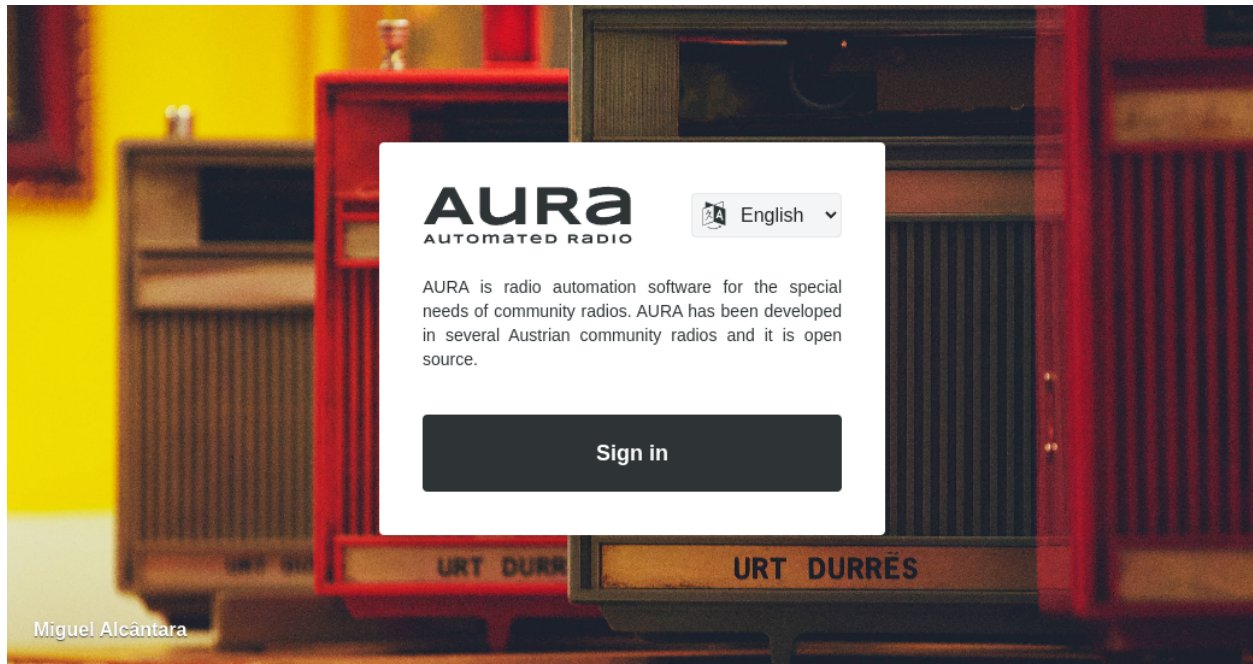
1.1.2 Login & logout

First choose the language you want Dashboard use with.

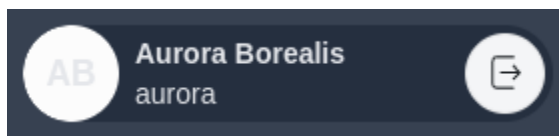


Then click the *Sign In* button to open your personal dashboard.

Use the credentials provided by your radio station.



After login you see your profile badge on the left top. Within that badge, click the icon on the right to sign out again.



1.1.3 Navigation

Sidebar

Use the navigation in the sidebar to navigate the Dashboard.

There are two main areas and their child pages:

- Shows: Displays a list of shows.
 - Episodes & Schedules: Here you can manage the broadcasting information of a specific show.
 - Show Settings: Configure the settings of the show here.
- Calendar: The calendar for programme planning.

Breadcrumb

Use the breadcrumb to quickly navigate to parent pages.

1.1.4 Inline editor for formatting text

Some fields feature an inline editor, allowing you to apply minor formatting. Those formatting options include:

- Bold text
- Italic text
- Strikethrough
- Bullet points
- Numbered lists
- Links

When entering URLs, they are automatically rendered as valid links. For your convenience you can even copy & paste pre-formatted text and its formatting is preserved.

To use the editor simply select the relevant text and a pop-over menu appears.

1.1.5 Automatic saving

Looking for save buttons? Search no more!

Any changes to form data is automatically transmitted to the server. When some automatic saving is in progress, this is indicated by a *saving...* icon, located next to the form element.



Changes affecting the appearance of Dashboard are saved to your browsers local storage. Next time you login to Dashboard they are applied automatically.

1.1.6 Information on data changes

Some pages display information about the date and time the record was created and when it was last updated. Depending on the page this information is located at the header or footer of the page.

1.1.7 Additional info in the footer

Scroll down to display the hidden footer of Dashboard.

The **Dashboard version** can be found on the far left of the footer. When reporting bugs you will be asked about the browser, its version and the current Dashboard application version in use.

Help & documentation links to the reference documentation you are currently reading. In the future we will provide a more seamlessly integrated and refined documentation option in Dashboard.

1.2 User and host profile

Dashboard differentiates persons and their data by applying two concepts:

- User
- Host profile

The chapters below explain their differences and relationship.

1.2.1 User

A user is identified by an account and a set of credentials, provided by administrators.

User roles

Some user account typically has one of these roles assigned:

- **Host / Editorial Staff** Typically a host is a person belonging to the editorial staff. Depending on the radio station a host has different permissions. Such permissions can allow full show administration or only certain rights on show or episode level. Hosts can also be only guests on certain episodes of shows they do not belong to (so called *contributors*). Such hosts have no login credentials and are mainly presented on the website description.
- **Programme Manager** A programme coordinator is the manager of a radio stations programme. It is in charge of organising the overall programme plan, involving schedules, consistency in show and episode information and other content quality assurance duties.
- **Radio Station Administrator** This is a superuser, typically part of the IT team. Radio Station Administrators are in charge of keeping the system up and running, performing updates and doing certain low level operations.

In addition to these roles, users can also have a set of individual permissions.

Users and their roles are configured in the *radio station administration*. In order to create users, assign roles or modify individual permissions, contact your administrator.

Edit personal user profile

To edit your personal user profile, click on the badge displaying your name on the left top corner.

#TODO: More content will be provided in a coming alpha release.

Edit multiple user profiles

As a programme manager you can edit user profiles of all users.

#TODO: More content will be provided in a coming alpha release.

1.2.2 Host profile

Hosts are public profiles of editorial staff members, as they are presented on the radio station website, the show profile etc. That means in most cases hosts are also users, and are referenced to each other.

Hosts can also be one-time guests of a show, also references to as *Contributors* in the episode details. Such guests can be assigned to an episode on-the-fly. They don't necessarily have a user account assigned and may not be active in the radio station community.

Host profile vs Host role

Note, that the described host profile is different to the host user role and must not be equally assigned. For example a user with the role *Programme Manager* can also be a *Host* of a show.

Below you learn more about managing your profile details or general host info, in case your are a programme manager.

Create new host profile

#TODO: More content will be provided in a coming alpha release.

Manage host profile

#TODO: More content will be provided in a coming alpha release.

Assign host to user account

#TODO: More content will be provided in a coming alpha release.

1.3 Shows and episodes

Learn about how to manage shows, their episodes and basic settings.

This is an essential resource for radio hosts and programme managers likewise.

1.3.1 List all shows

After logging in, you are presented with a list of available shows.

Depending on your role, you see a different set of shows:

- All shows (programme coordinator role)
- Shows you have administration rights for (host role)

Search and filter shows

Use the input box to filter for specific shows.

Enter a search term and the results are matched for show name and short description.

Order show list

By default the show list is ordered ascending by following criteria:

1. Current user is show administrator > not show administrator. This only applies to programme coordinators. In the *Card View* the results are segmented in “*My Shows*” and “*Other Shows*” blocks.
2. The show *is active* > *not active*
3. The slug of the show alphabetically A-Z

The *Order Shows* button allows you to modify the order the displayed results.

After clicking the button select the order criteria.

Click the checkbox for each order criteria item to activate/deactivate it. Use the buttons on the right to switch between *descending* and *ascending* order. To change the actual order of how the order criteria is applied, use the dotted handle on the far left.

Switch between list view types

The two buttons on the far right, allow you to change the appearance of the list.

You can switch between the *List View* and a *Card View*.

Change show details

After selecting a show from the list, you have the option to modify the basic show settings and its episode details in the *show planning area*.

1.3.2 Create a new show

To create a new show, click the button on the right top in the show list view.

Note, this button is only available for the programme coordinator role.

Next, fill out the form and submit.

If you are starting from scratch, you might not have any required *show type* and *funding category* metadata defined yet.

Learn how to manually define categorization options in the *Radio Station Administration* chapter.

Alternatively contact your IT support team, asking them to add a default set of metadata using fixtures.

Configure show details

After the show has been created, move to the *Show Settings* and update the default settings, like the show owner, categorization options, logo and other details.

Then switch to the *Calendar View* to create new schedules.

1.3.3 Show details

After a selecting an entry from the show list, you land on the **show details** page.

Show episodes

The first table shows a list of **future episodes** by default. The green badge on the right top of the table, indicates the episode playing next.

When clicking the *past* tab, you get a list of the **past episodes** which already aired.

Use the pagination in the table footer to see additional records. Enter a value in the input for **Episodes per page**, to change how many entries are listed.

Edit episode

Edit the episode by clicking within the first column of the table.

Enter a catchy *episode title*, a *summary* and a more detailed *description* on what is happening in this episode. The latter two fields allow basic inline formatting of the text.

Uploading an *image* can help getting additional attention for your episode.

The field *contributors* lists all *hosts* of the editorial staff by default. You can also add additional *contributors* for this episode, like some guests. Or remove some host, which is not participating in this episode.

Topics can be used to bundle a number of episodes and/or shows under a common theme. Provided *topics* can be configured in the *Radio Station Administration*.

Languages are inherited from the show settings, but can be customized per episode.

Tags provide the ability to add additional meta information to the episode. Its use can be defined to the liking of your radio station. Hit *Enter* to separate and save individual terms.

Add media sources

In the bottom area you can upload audio files and assign other types of media sources.

Any audio assign will be broadcasted during the episodes timeslot. If no audio is assigned, there are various mechanisms in place to avoid *dead air*.

Learn more about this in the *media management* section.

Show schedules

After selecting a show from the *show list* you now see the show details.

Scroll down to the bottom to see a list of show schedules.

By default you see the **current schedules**, meaning all the schedules which have not yet ended.

When clicking the *all* tab, you get a list of **all schedules**, including the ones from the past.

Create schedule

Switch to the *calendar view* to create new schedules.

1.3.4 Show settings

After selecting a show from the *show list* you are redirected to the show details.

In the sidebar navigation click on *Basic Settings* to reach the show settings page.

Here you can see and edit all the general settings of your show. Depending on your role, you may or may not be able to edit certain fields.

Basic information

Set the *show title*, *subtitle* and *description* here.

The *show title* is the basis for an automatically generated *slug*, required for radio website integrations. The *slug* can be altered in the *Danger Zone* below.

The *subtitle* is a tagline or short description, especially used to present the show in overview pages of the website. It allows basic inline formatting of the text.

The *description* is the place for your lengthy show description essay. It allows basic inline formatting of the text.

Content information

This section is used to assign content categorization options. All available data you can choose here, is predefined in the *metadata section of the radio station administration*.

The fields *topics* and *languages* can be overridden and extended on episode level.

Editors & contact information

Define contact information, like an *email* to allow people reach out to your show team. Depending on the radio this information may be only used privately, publicly or a mix of both.

You can define a set of links to be shown on the webpage of the show. The provided link types are defined individually by the radio station in the *radio station administration*.

The *editorial staff* is a set of people typically hosting the show. Their profiles are displayed as hosts on the show details of the radio station website.

The list of *administrators* define user accounts, which are allowed to login to the Dashboard and organize the show and their episode details.

Administrative information

The *funding category* is required for regulatory or funding purposes in certain areas. Available option can be freely defined in the *metadata section of the radio station administration*.

The CBA ID is a field currently not in use. It will provide integration of the *Cultural Broadcasting Archive (CBA)* at a later stage.

The *predecessor* can be used to refer to a historically related, preceding, but ended show. This is a special use case and can be ignored in most scenarios.

The *internal note* is a private text for *Programme Managers* to store notes on shows and their staff. Note this text can only be editing and read by *Programme Managers* and *Administrators*. It can be useful to store remarks on e.g. special contact options, info on sick leave of hosts, or anything required to ease the organizational work of *Programme Managers*.

Default media sources

In this section you can assign default media sources.

Any audio source assigned here is used, when there is no other audio source assigned on schedule or episode level. Learn more about the mechanics of assigning media sources in the *Media Management chapter*.

Danger Zone

The section holds operations to be handled with care, as they can result in non-revertable outcome.

All operations in the danger zone require you to confirm the action in an additional dialog.

Deactivate show

The *deactivate show* button is used to disable a show. When pressing the button, the show is marked as inactive and all existing, future timeslots are deleted. The editorial staff is not able to edit show and episode details anymore.

Delete show

The *delete show* button deletes the given show and all related data. This should be only used when you have created a show by mistake. Usually you want to deactivate a show instead of deleting it, since you want to keep historic show data to be potentially displayed on the radio station website.

Rename show slug

When renaming a show title, you might also want to update the show *slug*. Keep in mind, that this results in broken URLs on the radio station website. So check back with your IT team before renaming the *slug*.

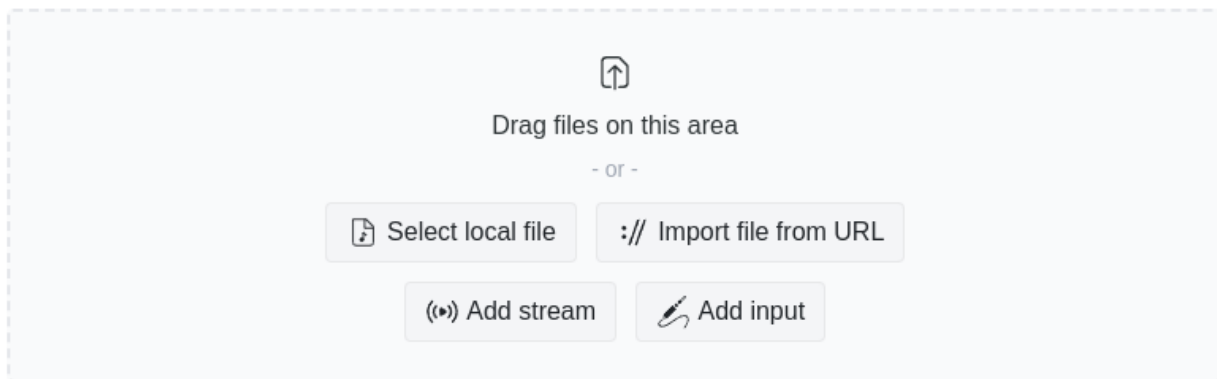
1.4 Media management

Learn how to upload audio files and organize various types of media sources.

1.4.1 Assign media source

Following media sources are allowed:

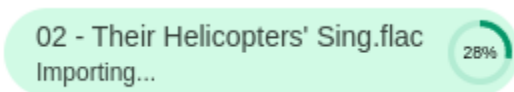
- Audio file upload from local computer
- Import audio file from a remote URL
- Add URL providing a live stream or stream from another website
- Line input, for broadcasting from the live studio



Local file

When you chose to upload a file from your local computer, you can directly drag & drop that file to the media source area.

The upload is starts automatically and it's progress is shown. After the upload step, your file is automatically normalized and converted to **FLAC** on the AURA server.



Import from URL

TODO: Add content & screenshot

Add stream

TODO: Add content & screenshot

Add input

TODO: Add content & screenshot

1.4.2 Alternative media sources

AURA provides multiple mechanisms to avoid *dead air*.

Default media sources

While you can set media sources on an episode level, there is also an advanced concept of assigning media sources on schedule and show level.

TODO: Add content & screenshot after UI changes

Fallback sources

Fallback sources are triggered by the in-built silence detector after a certain threshold of silence or noise is reached.

TODO: Add content & screenshot after UI changes

1.5 Calendar

The programme calendar is the home for programme managers to do their daily planning work.

Here you can get a quick overview on what the current programme looks like. This is also the place to create schedules and timeslots based on recurrence rules.

Dec 4 – 10, 2023

Today

< >

	Mon, 12/4	Tue, 12/5	Wed, 12/6	Thu, 12/7	Fri, 12/8	Sat, 12/9	Sun, 12/10
6:00 AM	6:00 AM - 6:30 AM Singing Birds	6:00 AM - 6:30 AM Christian in the Mix	6:00 AM - 6:30 AM on connaît la chanson	6:00 AM - 7:30 AM GRRRLS CLUB No playlist	6:00 AM - 6:30 AM canzoni italiane		
6:30 AM	6:30 AM - 7:30 AM Gen Z Nach Y kommt Z	6:30 AM - 7:30 AM Nerd Talk	6:30 AM - 7:30 AM Emigranti		6:30 AM - 7:30 AM Morgenrunde	6:30 AM - 7:00 AM alles was geht	
7:00 AM						7:00 AM - 8:00 AM Pretty Peggy-O	7:00 AM - 8:00 AM YUbox
7:30 AM	7:30 AM - 8:00 AM onda-info	7:30 AM - 8:30 AM VON UNTEN im Gespräch	7:30 AM - 8:30 AM Podcasts!	7:30 AM - 8:00 AM VON UNTEN	7:30 AM - 8:00 AM #Stimmlagen		
8:00 AM	8:00 AM - 9:00 AM c/o			8:00 AM - 9:00 AM Women on Air present: Globale Dialoge	8:00 AM - 8:57 AM Sunrise Orange	8:00 AM - 9:00 AM Fratts Selection	8:00 AM - 9:00 AM Klassik am Sonntag
8:30 AM		8:30 AM - 9:30 AM Radio Stimme	8:30 AM - 9:00 AM OHRENWARMER:				
9:00 AM	9:00 AM - 10:00 AM Selchfleisch		9:00 AM - 10:00 AM YUbox	9:00 AM - 11:00 AM Kulturradio	9:00 AM - 11:00 AM bum bum tschack x Gen Z	9:00 AM - 10:00 AM IndieRE 2.0	9:00 AM - 11:00 AM African Time
9:30 AM		9:30 AM - 10:00 AM Economic Update					
10:00 AM	10:00 AM - 11:00 AM Selchfleisch	10:00 AM - 12:00 PM Hotel Passage	10:00 AM - 11:00 AM GRRRLS CLUB No playlist			10:00 AM - 11:00 AM Jazz-News	
10:30 AM							
11:00 AM	11:00 AM - 11:55 AM GRRRLS CLUB No playlist		11:00 AM - 11:55 AM GRRRLS CLUB No playlist	11:00 AM - 12:00 PM Unerhört - Radio ohne Barrieren	11:00 AM - 11:55 AM Widewidewitt	11:00 AM - 12:00 PM Das rote Mikro	11:00 AM - 12:00 PM Crossing Borders
11:30 AM							
12:00 PM	12:00 PM - 1:00 PM Singing Birds	12:00 PM - 1:00 PM VON UNTEN im Gespräch	12:00 PM - 1:00 PM Podcasts!	12:00 PM - 12:30 PM VON UNTEN	12:00 PM - 12:30 PM #Stimmlagen	12:00 PM - 2:00 PM Indigenous Rights Collective Radio	12:00 PM - 2:00 PM Radio Rinia
12:30 PM				12:35 PM - 2:00 PM	12:30 PM - 1:30 PM		

1.5.1 Recurrence rules

Recurrence rules are the basis for creating schedules for a show. A recurrence rule defines the *repetition rhythm* a given schedule is based on.

Generally all of these recurrence rules can be made available:

- once
- daily
- weekly
- business days
- weekends
- bi-weekly
- four-weekly
- monthly on the first

- monthly on the second
- monthly on the third
- monthly on the fourth
- monthly on the fifth
- monthly on the last
- bi-monthly on the first
- bi-monthly on the second
- bi-monthly on the third
- bi-monthly on the fourth
- bi-monthly on the fifth
- bi-monthly on the last
- three-monthly on the first
- three-monthly on the second
- three-monthly on the third
- three-monthly on the fourth
- three-monthly on the fifth
- three-monthly on the last
- four-monthly on the first
- four-monthly on the second
- four-monthly on the third
- four-monthly on the fourth
- four-monthly on the fifth
- four-monthly on the last

The allowed rules differ from radio station to radio station. Your IT team will setup AURA with the rules relevant to your radio station.

Example for different supported recurrence rules:

- [Radio Helsinki](#), supports weekly, bi-weekly and four-weekly episodes.
- [Radio ORANGE 94.0](#), supports weekly, bi-weekly and monthly episodes.

Learn how recurrence rules relate to schedules and timeslots in the next chapter.

1.5.2 Organize schedules and timeslots

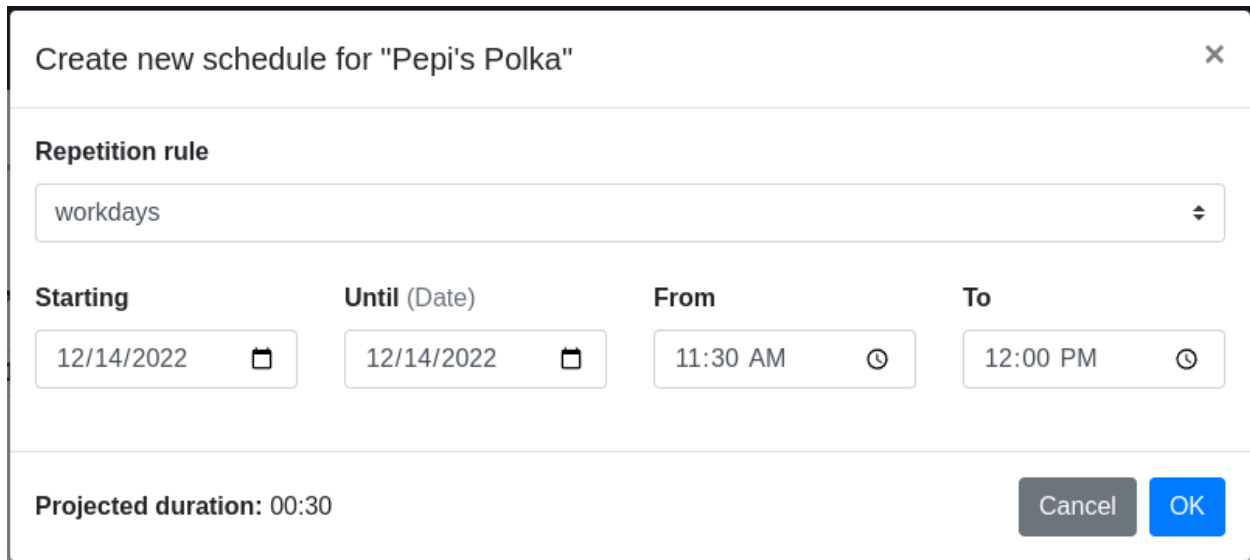
A show's schedule is defined by a start and end date/time. When creating a schedule a set of calendar entries, so called *timeslots* are created within that timeframe. These timeslots are used to assign episodes to be broadcasted.

As a result a set of calendar entries, so called *timeslots*, is created for the show's schedule.

Create a schedule

To create a new schedule, click the desired area in the calendar. The following dialog opens.

TODO: Add screenshot & description after UI changes in alpha-5

The screenshot shows a dialog box titled "Create new schedule for 'Pepi's Polka'" with a close button (X) in the top right corner. The dialog is divided into several sections. The first section is "Repetition rule" with a dropdown menu currently set to "workdays". Below this is a row of four input fields: "Starting" (12/14/2022 with a calendar icon), "Until (Date)" (12/14/2022 with a calendar icon), "From" (11:30 AM with a clock icon), and "To" (12:00 PM with a clock icon). At the bottom left, it says "Projected duration: 00:30". At the bottom right, there are two buttons: "Cancel" (grey) and "OK" (blue).

Select the recurrence rules matching your planning. Depending on the selected rule a different set of timeslots is created.

When creating a schedule, the calendar is able to detect collisions with existing, conflicting timeslots.

To learn more about this behaviour, check out the chapter below.

Update schedule

Click the timeslot to open its dialog.

TODO: Update screenshot & description after UI changes in alpha-5

Edit schedule for "Pepi's Polka"
×

This timeslot runs on **Wednesday, December 14, 2022** from **11:30 AM** to **12:00 PM**

This schedule repeats from **workdays** to **Thursday, December 15, 2022**

Start	End
Thursday, December 15, 2022 at 11:30 AM	Thursday, December 15, 2022 at 12:00 PM
Wednesday, December 14, 2022 at 11:30 AM	Wednesday, December 14, 2022 at 12:00 PM

«
<
1
>
»

Add repetition schedule

When to repeat?

On the following day
⌵

☒ Use same start and end time

Add repetition schedule

Fallback für Ausstrahlungs-Schema hinzufügen

Momentan hinterlegte Playlist: (keine)

Assign playlist

Delete schedule and all timeslots

Delete timeslot

Delete all future timeslots

Delete schedule

Click the timeslot to open its dialog.

On the bottom you find various options to either

- Delete all timeslots and its schedule
- Delete all timeslots in the future
- Delete the current timeslot only.

TODO: Update screenshot & description after UI changes in alpha-5

Delete schedule and all timeslots

Delete timeslot

Delete all future timeslots

1.5.3 Timeslot collision detection

When using the Dashboard Calendar to create new schedules or timeslots, there is some logic in place to avoid collisions of conflicting timeslots.

That means only a single timeslot is allowed for a single point in time.

When a collision is detected, the user interface offers you options to solve the conflict.

Single collision, fully overlapping

In this situation an existing and your newly created timeslot are fully overlapping.

Single collision, partly overlapping

Here only some duration of the new timeslot is overlapping an existing one.

Here you have the option to truncate your timeslot or the existing one. Additionally, the solutions offered in the previous scenarios are offered here too.

Single collision, superset and subset

In that scenario, conflicting timeslots are offered to be split. Again, all possible solutions from the previous scenarios are valid too.

1.6 Studio clock

The studio clock is a web application giving you an overview on:

- What is currently playing
- What is playing next
- The current time

`/*TODO: add content in following releases */`

1.7 Radio station administration

Radio station administrators have some additional features for radio management at hand. As a programme manager you have read-only access to this area.

You can reach that radio station administration interface by clicking the *Settings* button in the Dashboard navigation. This button is only visible to administrators and programme managers.

1.7.1 Radio station settings

Configure your station-wide settings in the admin area under *Radio Settings*.

General settings

Station name

Enter the official name of your radio station in the field.

Station website

Enter the main website here. This is not the URL where the AURA Dashboard is provided.

Station logo

Choose your radio station logo here.

Programme settings

Fallback show

The fallback show is used when the programme timetable holds empty timeslots.

In such cases AURA automatically creates *dynamic timeslots* assigned to the chosen show.

Typically this is a show, which is dedicated to random fallback music.

CBA settings

CBA API key

The CBA API key is used to translate any given CBA media source to the actual media file.

In the future it used for providing additional CBA API resources, for example to integrate AURA and CBA to your radio station website.

In addition CBA offers certain web customizations and analytics based on that API key. So it is important to use your personal radio station key.

Revisit this documentation in future releases of AURA and learn about advancements of the CBA integration.

Public API-use

Ensure not to use any internal CBA API key here. Since this is meant for public access, the given API key should not provide any advanced CBA features, like giving access to internal metadata or upload capabilities. When in doubt, check back with CBA if you are allowed to use the provided key here.

CBA domains

This JSON field holds an array of valid CBA domains.

These domains are used for automatically identifying CBA media links. Based on that, Dashboard is able to automatically translate such CBA URLs to actual media resource. Also compare the previous section on the CBA API key.

The default values are:

- `cba.media` - the current main CBA domain.
- `cba.fro.at` - the legacy CBA domain, still kept for compatibility purposes.
- `cba.aura.radio` - replace this with `cba.myradio.org` if your radio provides a CBA proxy.

Play-out settings

Line-in channels

This JSON mapping holds a list of all available studios at your radio station.

Add all studios which are connected to AURA Payout via line-in.

This is the basis for displaying available line-in channels in the media source section of Dashboard.

The key of an entry holds the channel number, starting with 1. Ensure, that each channel you are defining here, there needs to be an equivalent configuration in AURA Payout. The value of each is the channel name, how it will be presented in Dashboard.

1.7.2 User roles and permissions

There are three default roles

- **Host / Editorial Staff**
- **Programme Manager**
- **Radio Station Administrator**

To learn more about these roles, check the [user and host profile](#) page.

In the radio administration these roles are realized using groups and an assigned set of permissions. But any of these permissions can also be assigned individually to user accounts.

The *Host / Editorial Staff* role is provided in two flavors, in form of a host **Host** and **Host+** group. This allows administrators to get an impression on different permission capabilities more quickly. When finalizing the radio station configuration, decide for one of these *Host* groups and modify individual permissions to your organizational needs.

Public visibility of data

Whenever editing any data, keep in mind, that almost all data is visible via public APIs. All the data is read-only for non authenticated users. Basically the API provides all data which is typically also visible on radio station websites.

Additionally, this also includes metadata like when and by whom a record was created or updated last.

The only exceptions are those fields, which are only visible internally and only editable by certain roles:

Area	Field	Visible only to these roles
user	first name	All authenticated users
user	last name	All authenticated users
user	email	All authenticated users
host	email	All authenticated users
show	email	All authenticated users
show	internal_note	Programme managers

Open data principle vs. user privacy

As advocates for open data, we're all about transparency, just like community radio stations. But remember, always follow the law in your area. For example, in the European Union (GDPR), users need to agree before you share their personal data. So, be sure to get their written consent first.

Secure storage of user passwords

We want to assert, that passwords are only stored in hashed form and are therefore not even visible to administrators. They are never exposed by API, since authentication is performed using OpenID.

Groups and their default permissions

Beside the default read-access, the groups are pre-configured with these permission assignments:

Area	Field	Host	Host+	Programme Manager
show	name			edit
show	slug			edit
show	short description		edit	edit
show	description		edit	edit
show	logo		edit	edit
show	image		edit	edit
show	categories			edit
show	topics			edit
show	music genres			edit
show	languages			edit
show	type			edit
show	email		edit	edit
show	links		edit	edit
show	hosts / editorial staff		edit	edit
show	administrators			edit

continues on next page

Table 1 – continued from previous page

Area	Field	Host	Host+	Programme Manager
show	funding category			edit
show	cba id			edit
show	predecessor			edit
show	internal_note	<i>no read access</i>	<i>no read access</i>	display, edit
show	is active			edit
show	default media source		edit	edit
schedule	default media source		edit	edit
episode	title	edit	edit	edit
episode	summary	edit	edit	edit
episode	content	edit	edit	edit
episode	image	edit	edit	edit
episode	contributors	edit	edit	edit
episode	topics			edit
episode	languages		edit	edit
episode	tags	edit	edit	edit
episode	links		edit	edit
media-source	file	select	select	select
media-source	line		select	select
media-source	stream		select	select
media-source	import		select	select
host	name	edit	edit	edit
host	biography		edit	edit
host	email		edit	edit

To serve the overview, permission strings in the table are simplified. When looking up the actual permission string, take the permission verb and combine it with the field name in format `Can {verb} {field}`. For example `Can edit music genres` is the permission string you will find in the permission assignment interface.

All cells without a permission string represent read-only access, except the ones marked with *no read access*.

Each group also holds wildcard permission assignments like `add episode`, `change episode`, `delete episode` and `view episode` which is not visible here. These are required for properly granting access to data by inheriting global permissions. Some are required for permissions through ownership (see next chapter). Generally you should not edit these assignments.

Permissions through ownership

Permissions through ownership only affects the *Host* and *Host+* roles, since *Programme Managers* have all relevant permissions to manage all shows and profiles.

Ownership of shows and episodes

Users which are assigned as *Show Admins* in the show settings, automatically have the permissions to list and edit their own shows, according to the assigned role (*Host*, *Host+*, or individual permissions). The permissions listed in the table, only grant users editing rights for shows they are actually owning.

Ownership permissions on shows and their episodes is granted by having these wildcard permissions assigned:

Can change show
Can change episode

Ownership of host profiles

In case of the host profile, only the owner can edit the fields, they have permissions on. Here ownership is represented by a relation between user and the host profile.

Additionally editorial staff members need the ability to assign guests to certain episodes (contributors). To do so, they often need to create host profiles on-the-fly.

Ownership permissions on host profiles is granted by having these wildcard permissions assigned:

Can change host

Assign roles and permissions

Read here, to learn on how to assign roles and permissions:

- [Manage users](#)
- [Manage groups](#)

1.7.3 Manage users

Before proceeding, learn more about *roles and permissions*.

List users

To get a list of available users click “*User*”.

From here you can click individual users for editing their details. It is also possible to search or filter users by different attributes.

Create user

To create a new user, click the button “*Add user*”.

LDAP creates users automatically

Please check back with your IT team, if you are having LDAP in place. When using LDAP there is no need to create users manually.

Enter user credentials

On the first form you need to enter a username and password, then click “*Save and continue editing*”.

Provide personal info

Then complete provide personal information, like their name and email address.

Set permission flags

Then, under *Permissions*, ensure the checkboxes *Active* and *Staff status* are checked. These ensure the user can actually login to the Dashboard.

Superuser checkbox

Checking this flag, assigns the *radio station administrator* role and equips the user with all available permissions. So handle this checkbox with utmost care.

Assign groups

After this, set the user role by assigning them to one or more groups.

Ensure you have read the *roles and permissions* overview first.

Host+ role inheritance

If you want to assign the *Host+* group, you also have to add the *Host* group. The reason is due to some internal inheritance logic. In case of the programme manager role you only have to assign the single group.

Assign individual permissions

Alternatively, you can assign additional, individual permissions to the user.

Ensure you have read the *roles and permissions* overview first.

After adding relevant permissions click “Save”.

Deactivate and delete user

The administration panel allows you to delete users by pressing the “Delete” button.

Alternatively, you can also de-select the *Active* or *Staff status* checkbox. This means the user data is still available, wherever it’s needed, but the user is not able to login anymore.

Always prefer deactivation over deletion

When in doubt which approach you should choose, always prefer to deactivate the user account. Deleting a user destroys all the historic data and should only be chosen when you are aware about the consequences.

1.7.4 Manage groups

Before proceeding learn more about *roles and permissions*.

Modifying groups should be planned with the radio managers and IT administration, since groups and permissions are usually carefully planned and defined within the whole organisation.

List groups

Click *Groups* to get a list of available groups.

Edit group

To edit the group click the group name in the list.

In that user interface you can change the group name or edit the set of permissions for that group.

Add group

To add a group click the button *Add group*.

1.7.5 Manage metadata

There are different kinds of metadata which can be used to attribute shows.

Categories

Categories can be utilized to create sections within the radio programme.

For example Radio ORANGE 94.0 uses categories to group their shows in form of *areas of interest* on their website.

Funding categories

Funding categories are used to mark shows as per funding-related and regulatory purposes.

Languages

Languages can be assigned on show level.

Music focus

Here you can define a list of music genres and assign them to shows.

Topics

This a way to group shows under one theme or special programme (German: “Schwerpunktprogramm”).

Types

Types can be used to define the format of the shows. For example you can mark shows as a *feature* or *talkshow*.

The following image gives an idea how Radio Helsinki uses this metadata group on their website.

Link Types

Link types define which types of links can be added to show, episode and host profile pages.

You can freely add new link types. If you want to prohibit the use of an already used link type, it is important to deactivate it, instead of deletion. Otherwise those links will not be properly displayed on the radio station website anymore.

1.7.6 Generate reports

Most radios require the generation of annual reports for regulatory purposes.

Reporting feature in Dashboard

At the moment reports have to be compiled by some administrator. In the future Dashboard will provide a glamorous [button to generate reports](#).

Schedule reporting

At the moment not native reporting capability is provided. Individual radio station have their own database query script to generate reports. In the future we will provide basic and advanced reporting mechanisms.

Play-out reporting

To generate playlog reports you can use the [playlog endpoint](#) of Engine API. These reports have to be manually generated and aggregated for now.

Learn how playlogs are structured in the API Schema for [PlayLog](#).

ADMINISTRATION GUIDE

Learn how to make and keep AURA playful.

2.1 Overview & Features

The following diagram outlines the main elements of AURA.

2.1.1 The naming scheme

Automated Radio, AutoRadio, AuRa or AURA, follows an “*Automobile-naming-scheme*”. All the components are named after some crucial parts of a car:

- **Steering:** This is the single source of thruth, holding all the radio data, hence steering some radio’s broadcast.
- **Tank:** Just spleen without steam. That’s where your shows, tunes and recordings are managed. Fuelling your broadcast with materials.
- **Dashboard:** Observe and control what’s happening. That’s the backend user interface of your radio with individual views for hosts and programme managers.
- **Engine:** The playout server allowing you to broadcast via FM and web audio streams. Includes silence detection and optional recording options.
- **Play:** Every car has some fancy *Auto Radio Player*. And every radio station has the need for some modern frontend experience. *Play* is a library of web components. Making web development a breeze. Just like *playing with blocks of Lego*.

2.1.2 Features

The aforementioned naming scheme is used as a basis for name-spacing individual services.

Here you get an overview of the features of the provided services.

Find details on additional functionality on the *bundle documentation* pages or in the configuration files of individual services.

Steering

`/* to be defined */`

Dashboard

`/* to be defined */`

Dashboard Clock

Web Application providing a Studio Clock.

`/* more to be defined */`

Tank

`/* to be defined */`

Tank Cut & Glue

`/* to be defined */`

Engine

Scheduling and control for the playout.

- **Scheduler:** Automatically broadcast your radio programme (see [AURA Dashboard](#) for a user interface to do scheduling)
- **Autonomous playout:** Schedule information is pulled from Steering into a local cache. This way the playout keeps working, even when the network connectivity might be lost.
- **Versatile Playlists:** Playlists can contain different content types, such as audio files, audio streams and line in channels of you audio interface.
- **Default Playlists:** Define playlists on show and schedule level, in addition to any timeslot specific playlists. This way you always have an playlist assigned, when some host forgets about scheduling a specific programme.
- **Heartbeat Monitoring:** Frequently send pulse to a monitoring server, ensuring your schedule and playout server is up and running.

`/* more to be defined */`

Engine Core

The playout server based on Liquidsoap.

- **Multi-channel input:** Play audio from various sources including queues, streams and line-in from the audio interface
- **Multi-channel output:** Output to line out or audio streams
- **Icecast connectivity:** Stream to an Icecast Server, provide different encoding formats and bitrates

- **Auto DJ triggered by Silence Detector:** Play fallback audio triggered by a silence detector to avoid *Dead Air*. Play randomized music from a folder or M3U playlist.
- **Metadata handling:** Send information on playlogs via REST calls to external services like Engine API.
- **ReplayGain:** Normalization done using passed *ReplayGain* meta data.

Engine API

An OpenAPI 3 service to store and retrieve Engine data.

/ more to be defined */*

Engine Recorder

A simple but powerful recorder.

- **Bulk Recorder:** Record blocks of audio for archiving, audit-logging or further processing
- **Sync to archive:** Periodically synchronize recordings to a defined destination

2.2 Migration & Integration Plan

Before getting started you need to carefully plan how you want to fit AURA into your infrastructure.

Check out the provided *Deployment Scenarios* on ideas for integration AURA into your ecosystem.

2.2.1 Individual test & production instances

To keep your radio infrastructure up-to-date, AURA is meant to provide frequent releases.

In order to keep this process free of glitches, it's good to gain experience in maintaining AURA on a test environment.

But also after moving AURA to production it is highly recommended to have a staging- or test-instance, where new releases are deployed before actually updating the production servers.

2.2.2 Migration Plan

Rome wasn't build in a day. And a radio software ecosystem isn't migrated in one day either.

Make a plan with different stages. Decide which parts you want to migrate at a certain stage.

Possible stages are:

1. Migrate and continuously synchronize scheduling and show data
2. Integrate your website and other frontends with AURA
3. Test run the actual play-out
4. Make the switch to the new play-out system

2.2.3 Migration Tools

Several radios are building tools and scripts for migration. Besides offering Best Practices, we can also share some code for a sample migration service, if that's what you need.

Let's talk in [Matrix](#).

2.3 Deployment Scenarios

AURA comes with two Docker Compose bundles:

- **AURA Web:** Combines all essential containers required for the web-facing elements.
- **AURA Payout:** Everything related to scheduling, play-out and (optionally) recording.
- **AURA Record:** A standalone recorder and archive synchronisation.

There are also additional, individual Docker containers available to further extend the provided features.

The following network diagrams outline ideas on deployment scenarios.

2.3.1 Single Instance

This is the most simple scenario for setting things up, as all services reside on the same machine. The downside is, that any hardware issues or the need for restarting the server has an immediate effect on the full array of services.

2.3.2 Advanced

Here we utilize at least two machines for services: One for all web-facing applications (AURA Web), the other related to scheduling and play-out (AURA Payout). This is the most versatile approach between simplicity and maintainability, since your play-out will still be functional, while shutting down the web server for updates.

2.3.3 High Availability

That's the most sophisticated approach where your radio should never ever go down. But it's also the most tricky one, which needs careful testing and more valves to maintain.

Future scenario

Please note that this is a future scenario. Not all required service are available yet.

2.4 Deployment Preparations

2.4.1 Requirements

- Git
- make
- Docker Engine 25.0.2 or later
- Docker Compose 2.15.1 or later

2.4.2 Choose a domain name

First decide where you want AURA to be served to your users. Given your radio website is available under `myradio.org`, we recommend binding AURA to the subdomain `dashboard.myradio.org`. Dashboard is the application you will interact most often with, therefore it shall get a prominent place in your URL scheme.

Update your DNS settings with your desired domain name and point it to the server instance you want to deploy AURA at.

In all of the following examples, we will refer to the `dashboard.myradio.org` example. Replace it with your personal AURA subdomain, where applicable.

2.4.3 Setting up the user & home directory

It's time to log in to your fresh AURA server instance and get things moving.

Clone the aura repository to your desired AURA home directory.

We recommend using `/opt/aura` for the AURA installation.

```
$ sudo mkdir /opt/aura
$ sudo chown -R $(id -u):$(id -g) /opt/aura
$ git clone https://gitlab.servus.at/aura/aura.git /opt/aura
```

Next, move to that directory and create the aura user and group:

```
$ make aura-user.add
```

This is user merely required for handing out correct permissions and ownership. It is not meant to login with.

Current user vs. aura user

All operations are meant to be done by the current user. This user is also added to the aura group, hence acts on behalf of the aura user.

2.4.4 Docker post-installation steps

Install *Docker Engine* to be managed as a non-root user. Choose the previously created aura user to be in charge of running your containers.

Also, you don't need any fancy additions like *Docker Desktop*.

Ensure that Docker default context is enabled

Docker Desktop may change the current context. Switch back to the default Docker context with `docker context use default`.

Docker in production

When configuring the Docker Engine for *production*, you may want to start all your containers automatically on system boot.

This can be achieved by running Docker Engine as a daemon. We recommend using the aura user for running Docker Engine using `systemd`.

2.4.5 Selecting the release

Ensure you have the latest codebase:

```
$ git pull
```

Then check for available releases:

```
$ git tag
1.0.0-alpha1
1.0.0-alpha2
1.0.0-alpha3
```

This command returns a list of all tags corresponding to the available releases. Alternatively you can take a look at [releases.aura.radio](#).

To learn about changes between releases, consult the [Release Notes](#).

Then switch to the release you'd like to use.

```
$ git checkout tags/<release-version>
```

Replace `<release_version>` with one of the versions listed above, like `1.0.0-alpha1`.

```
$ git checkout tags/1.0.0-alpha1
```

Latest, unreleased state on the main branch

In case you want deploy the current development state, just take the latest commit on the `main` branch. That's the state which is checked out by default. While we try to keep the `main` branch as stable as possible, we cannot guarantee it being functional at all times. So please use it at your own risk.

2.4.6 Setting up the Audio Store

The *Audio Store* is a folder which is utilized by *Tank* and *Engine* to exchange audio files.

Assuming both, *Engine* and *Tank* are hosted on different machines, audio folders must be shared using some network share.

In case you are hosting *Engine* and *Tank* on the same machine, you can skip this step. Just think about pointing the settings the relevant audio directories, or create a symlink to do so behind the curtains.

By default the audio store is located in `/opt/aura/audio`. There are following subdirectories expected:

- **source:** Holding all audio files from the media asset repository. Written by *Tank* and read by *Engine*.
- **fallback:** Populate this folder with audio files to be played randomly, in cases where nothing is scheduled.
- **playlist:** Used for M3U audio playlists.
- **recordings:** The recorder stores its recorded blocks here.
- **import:** To import audio files into Tank via the filesystem, place them here.

Share Type

Then, there's the question how the share is managed. Feasible options include:

- NFS
- SSHFS
- Gluster

Please evaluate for yourself what the most failsafe solution is. The next chapter outlines pro and cons of different scenarios.

Share Location

You have following options where your share can be located:

1. **Engine and all other AURA components (Tank, Dashboard, Steering) are running on the same instance.** This is the most simple solution, as Engine and Tank can share the same directory locally. But this scenario requires some more sophisticated tuning of the system resources to avoid e.g. some overload of multiple Uploads in Tank may affect the performance of engine. You can eliminate this risk by setting CPU and memory limits for Steering, Dashboard and Tank using Docker or `systemd-cgroups`. A disadvantage here is the case of maintenance of system reboot. This would mean that all components are offline at once.
2. **Physical directory where the Engine lives, mounted to Tank.** This may cause an issue with the mount, when the network connection to Engine is unavailable or the instance is rebooting.
3. **Physical directory where the Tank lives, mounted to Engine.** This may cause an issue with the mount, when the network connection to Tank is unavailable or the instance is rebooting.
4. **Central Data Store or Storage Box** which is mounted to Engine and Tank. In this case a downtime of the store make both, Engine and Tank dysfunctional.
5. **Replicated storage solution using Gluster, both Engine and Tank have their virtual audio directory mounted.** That's the ideal approach, because if any of the instances is down, the other has all the data available.

In any case, you should think about some backup solution involving this directory.

Learn more on how what should be backed up in the chapter [Update and Maintain / Backup Strategy](#).

2.4.7 Deployment

AURA can be deployed using Docker and Docker Compose, allowing custom-tailored orchestration.

In the next chapters you learn how to deploy the individual Docker Compose bundles:

- AURA Web
- AURA Playout
- AURA Recorder

2.5 Deployment Bundles

Same as a car has fixed and moving parts, AURA likewise has parts which are fundamental and parts which are optional.

In order to keep administrators life simple, we have defined component spaces reflecting to most common deployments:

- AURA Web: Contains all the web facing services, including digital asset management
- AURA Playout: Contains all things required for scheduling & broadcasting
- AURA Record: Record audio and periodical sync to archive

These spaces are realized as of [Docker Compose](#) bundles.

2.5.1 AURA Web

Service	Required	Description
steering		Single-source of truth, holding all radio, show, hostand scheduling data
tank		Upload, download, normalization and media asset management
dashboard		Backend user interface
dashboard-clock		Studio clock
play		A library for building frontends based on web components

The *required* services are needed for minimal functionality.

2.5.2 AURA Playout

Service	Required	Description
engine		Control and scheduling for the play-out
engine-core		Play-out Engine
engine-api		API for playlogs and track service
engine-recorder		Record audio blocks, including archiving

The *required* services are needed for minimal functionality.

2.5.3 AURA Recorder

Service	Required	Description
engine-recorder		Record audio blocks, including archive sync

The *required* services are needed for minimal functionality.

2.6 AURA Web

For all of the following steps, please change to `/opt/aura`, also referenced as `$AURA_HOME`.

2.6.1 Quick Installation Guide

Deployment preparation steps:

```
$ git pull
$ git tag
  1.0.0-alpha1
  1.0.0-alpha2
  1.0.0-alpha3
$ git checkout tags/<release-version>
$ make aura-user.add
$ make aura-web.init
```

Modify the newly created *configuration file* and initialize the database:

```
$ nano config/aura-web/.env
$ make aura-web.configure
```

Now create *fixtures* based on samples, edit or delete files. Then import fixtures as the last step:

```
$ make aura-web.fixtures-create
$ nano config/aura-web/fixtures/...
$ make aura-web.fixtures-import
```

After this is set, start AURA Web with:

```
$ make aura-web.up
```

Watch the health status and logs with:

```
aura$ docker ps
aura/config/aura-web$ docker compose logs -f
```

For additional maintenance options, see *Update and Maintenance*.

2.6.2 Complete Installation Guide

Call following command to initialize the Aura Web environment:

```
$ make aura-web.init
```

This command does

- create a configuration file `.env`, based on the sample configuration `sample.env` located in `config/aura-web`.
- update the ownership and permissions of relevant files and folders.
- create a set of sample fixtures located in `config/aura-web/fixtures`. It's up to you, if you want to use them.

Modify the configuration file

The previous step already created a `.env` file based on `sample.env`.

Now let's configure the essentials.

Configure host and certificate generation

Now you need to configure how AURA is reachable.

Environment variable	Values (default)	Description
AURA_HOST_NAME		Set a LAN host or a domain name. See below why to avoid <code>localhost</code> .
AURA_HOST_PROTO	https , http	Defaults to https . Set to http if you are running locally.
AURA_HOST_CERTBOT_ENAB	true , false	Set to false if you are running locally or retrieve certificates differently.
AURA_HOST_CERTBOT_EMAIL		Set a valid email for certbot.

By default it is meant to be reached via a publicly facing domain name. AURA by default offers automatic generation of TLS certificates using [certbot](#).

Therefore you only need to set the values for `AURA_HOST_NAME` and `AURA_HOST_CERTBOT_EMAIL`.

Ensure you use a fully qualified domain name, like `dashboard.myradio.org`, and that your firewall allows outside connections from port **80** as well.

Changing the host

Any update of host and protocol in your `.env` file later on, is not reflected in the actual configuration. In such cases you either need to manually update the relevant OIDC client database tables, or you simply create new OIDC client IDs in the configuration. After that you can delete any old OIDC clients via the Steering admin interface `/steering/admin/oidc_provider/`. Read more about this in the [Update & Maintain section](#).

Local deployment

When deploying AURA locally, you can choose a LAN hostname. For example you can define `aura.local` in `/etc/hosts`. Then set this value for `AURA_HOST_NAME`. Also ensure you use `http` as protocol and to disable `certbot`.

Avoid using localhost and 127.0.0.1

Since the hostname `localhost` and the IP `127.0.0.1` is ambiguous when using Docker, ensure to avoid these in your configuration altogether. Otherwise this will result in a dysfunctional installation.

Configure connectivity to other services

Depending on your infrastructure setup, also set these environment variables:

Environment variable	Description
<code>AURA_AUDIO_STORE_SOURCE</code>	Tank stores audio files here. It defaults to <code>aura/audio/source</code> .
<code>AURA_TANK_ENGINE_PASSWORD</code>	Choose a password and remember it for configuring AURA Payout.
<code>AURA_ENGINE_API_INTERNAL_URL</code>	Set URL if AURA Payout is deployed to another host.

Apply the database configuration

To initialize the database based on the configuration file, call the following:

```
$ make aura-web.configure
```

This step will

- Create the database by running the migrations,
- Create the RSA Key
- Create the OpenID Connect clients for `dashboard` and `tank`,
- Create a super user.

Import default radio station metadata

The radio station metadata settings can be imported using so-called *fixtures*. Fixtures hold a set of default or sample data, which can be imported, avoiding manual creation of certain records.

To create a set of fixtures based on sample data call:

```
$ make aura-web.fixtures-create
```

This creates a set fixtures in `config/aura-web/fixtures` based on the samples provided in `config/aura-web/fixtures/sample`. Note that any existing file are overwritten.

The following table gives an overview on fixture types and their use.

File	Re-quired	Description
category.json		Metadata to categorize shows with.
fundingcategory.json		Metadata for reporting on shows. Required for creating new shows.
host.json		Allows creation of a default host, assigned to a default show.
language.json		Supported languages assigned to shows or episodes.
license.json		Supported licenses used for image or media uploads.
linktype.json		Types of links available for shows, episodes or host profiles.
musicfocus.json		Music genres for classifying music shows.
radiosettings.json		Basic radio station configuration.
rrule.json		Recurrence rules used in planning of the radio station calendar.
show.json		Allows creation of a default show, like for filling empty timeslots.
topic.json		Metadata for grouping shows and episodes per certain topics.
type.json		Metadata for grouping shows per content format. Required for creating new shows.

Edit or delete individual files, depending on your needs. Most types of fixtures can be skipped, but there are certain required ones, needed for a functional user interface. If you only want to give AURA a try and have some starting points, it makes sense to import all data, as is.

After reviewing and adapting fixtures to your radio station needs, execute following command to import them to the database:

```
$ make aura-web.fixtures-import
```

Add and edit metadata via administration user interface.

Do not worry if you forgot to define some default records. You can also add and modify these metadata records in the [Radio Station Administration](#).

Start the services with Docker Compose

To start all services defined in the `docker-compose.yml` file, run:

```
$ make aura-web.up
```

Containers which are not yet existing are created. If some container is already running and is changed in some way, Docker Compose re-creates it.

This deployment will make *AURA Web* reachable in the following way:

Service	URL	Description
Dashboard	/	Dashboard is reachable directly \$AURA_HOST_NAME.
Steering	/steering	Steering holds all the programme data.
Tank	/tank	Tank is the uploader or audio repository.
Track Service API	/engine/api/v1/ trackservice	Track Service API is only available when engine-api is running.
Icecast	:8000/icecast	Available when optional Icecast server for the reverse proxy is enabled.
Dashboard Clock	/clock	Only available in LAN and when enabled in reverse proxy config.

Advanced configuration (LDAP and others)

If you need to extend the settings for production beyond what is possible with environment variables, you will need to create an additional configuration file for Steering. First copy the the provided sample configuration to the services folder:

```
$ cp config/services/sample-config/steering.py config/services/
```

Also compare the chapter on [Advanced Configuration](#)

Configuring LDAP

TODO: Explain LDAP Configuration, see <https://gitlab.servus.at/aura/aura/-/issues/289>

Start with advanced configuration

At the moment you cannot use the usual `make aura-web.up` to start with advanced configuration. Instead you need to use the following command:

```
$ config/aura-web/$ docker compose -f docker-compose.yml -f steering-production.yml up -d
```

This behaviour will be simplified in a coming release.

2.7 AURA Payout

2.7.1 Prerequisites

- ALSA compatible audio interface.
- [PipeWire](#) including *WirePlumber* installed on the host machine.
- In addition you will need the Pipewire JACK plugin (`pipewire-jack`).
- In production, we recommend Pipewire running under the aura user space.

Check if PipeWire is running

Before proceeding if, check if PipeWire is running with `systemctl --user status pipewire.service`.

2.7.2 Deploy AURA Payout

To initialize the payout deployment execute:

```
$ make aura-payout.init
```

This creates a configuration file `config/aura-payout/.env` based on the `sample.env` located in the same directory. Additionally this command creates the required folder structure and updates relevant permissions.

Update the configuration file

For a production deployment verify at least the following settings. For testing the payout, the defaults should work in most cases.

Environment variable	Description
AURA_TANK_ENGINE_P	The password should match the one configured in AURA Web. Avoid using the default one.
AURA_AUDIO_STORE_S	The location where Tank is storing audio sources. It defaults to <code>audio/source</code> and points to the one set in AURA Web.
AURA_AUDIO_STORE_P	The location where M3U playlists are provided. This is optional and defaults to <code>audio/playlist</code> .
AURA_AUDIO_STORE_F	The location where fallback audio is retrieved from. Such audio is played when nothing is scheduled. It defaults to <code>audio/fallback</code> .
PIPEWIRE_USER_ID	The UID of the user running the PipeWire server. Usually this is the current user (<code>id -u</code>).

In case you prefer deploying AURA Web and AURA Payout on two distinct server instances (e.g. *Advanced Deployment Scenario*), you have to update the following variables too.

Environment variable	Description
AURA_STEERING_BAS	Points to Steering in the local Docker network by default. Change value to <code>https://dashboard.myradio.org/steering/</code>
AURA_TANK_BASE_UR	Points to Tank in the local Docker network by default. Change value to <code>https://dashboard.myradio.org/tank/</code>

It is important to note, that the URLs have to end with a trailing slash (/).

Verify if AURA Web is running

In order to avoid complicated debugging of the connection between AURA Payout and AURA Web, try if the URL `https://dashboard.myradio.org/steering/admin` is working. It works if you get a login page served.

Provide some audio files for fallback

Now fill the folder `audio/fallback` with some music, or the directory you have set for `AURA_AUDIO_STORE_FALLBACK`.

Any audio files located here will be picked up and played when nothing else is scheduled.

Connect AURA Playout with your audio interface

There are three options to wire the playout with your audio interface:

- *Option 1:* Use an UI tool like `qpwgraph`, simply draw connections between the desired devices
- *Option 2:* Use the cli tool `pw-link`
- *Option 3:* Use the auto connect feature

When connecting the correct audio ports you will hear the fallback music playing immediately.

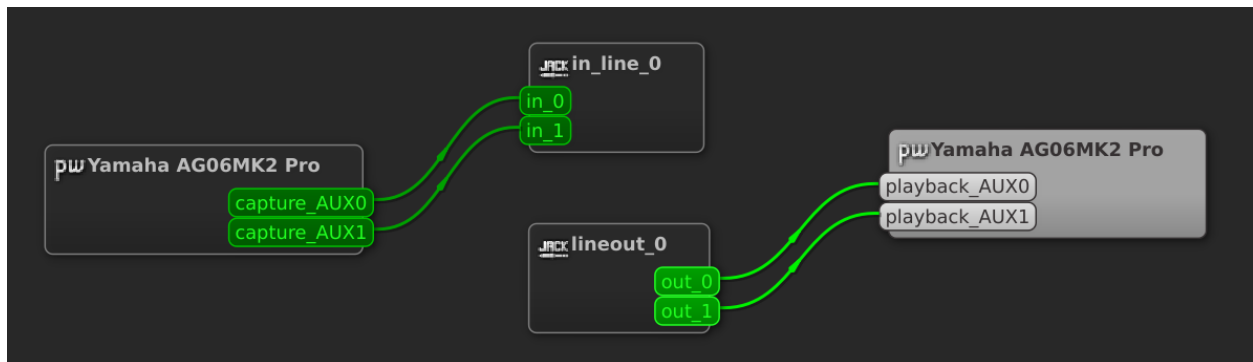
We recommend option one or two to start with, since the third variant can be a bit error-prone. We will improve on these options in the future.

Option 1: UI tool

First, start *AURA Playout* with

```
$ docker compose up -d
```

Then open a graph manager compatible with PipeWire, like `qpwgraph`. Now simply connect the *AURA Playout* nodes `in_line_0` and `lineout_0` with your audio device.



Option 2: CLI tool pw-link

To connect *AURA Playout* with your audio device(s) start everything with

```
$ docker compose up -d
```

Now list your input and output ports. There is a port for each channel on your device and a port for the left and the right channel from *AURA playout*.

```
# List the input ports
pw-link -i
# List the output ports
pw-link -o
```

Connect your port like in the example below, where you replace `alsa_input.usb-Yamaha_Corporation_Yamaha_AG06MK2-00.pro-input-0:capture_AUX0` with your audio device given from `pw-link`.

```
# An example for connecting everything:

# line out
pw-link "lineout_0:out_0" "alsa_output.usb-Yamaha_Corporation_Yamaha_AG06MK2-00.pro-
↪output-0:playback_AUX0"
pw-link "lineout_0:out_1" "alsa_output.usb-Yamaha_Corporation_Yamaha_AG06MK2-00.pro-
↪output-0:playback_AUX1"
# line in
pw-link "alsa_input.usb-Yamaha_Corporation_Yamaha_AG06MK2-00.pro-input-0:capture_AUX0"
↪"in_line_0:in_0"
pw-link "alsa_input.usb-Yamaha_Corporation_Yamaha_AG06MK2-00.pro-input-0:capture_AUX1"
↪"in_line_0:in_1"
```

Option 3: Auto connect feature

If you don't want to run Option 1 or Option 2 every time you start *AURA Playout* there is a way to connect everything when you start *AURA Playout*. For this you need to set the audio device(s) and channels in the `.env` file before you start *AURA Playout*.

```
$ docker compose run engine-core wpexec /etc/wireplumber/scripts/ls-ports.lua
```

This dumps details on every connected audio device. Grab the `port.alias` of your device and enter it in the `.env`. Press `CTRL + C` to exit the script.

Example Configuration

Given this device information, dumped by the previous command:

```
{ ["audio.channel"] = AUX0,
  ["port.physical"] = true,
  ["port.alias"] = Yamaha AG06MK2:capture_AUX0,
  ["node.id"] = 61,
  ["format.dsp"] = 32 bit float mono audio,
  ["port.direction"] = out,
  ["port.name"] = capture_AUX0,
  ["port.terminal"] = true,
  ["object.serial"] = 324,
  ["object.id"] = 66,
  ["object.path"] = alsa:pcm:1:hw:1,0:capture:capture_0,
  ["port.id"] = 0,
}
```

Set these values for the input and output settings in your `.env` file:

```
# Audio Device Settings
AURA_ENGINE_OUTPUT_DEVICE=Yamaha AG06MK2:playback
AURA_ENGINE_OUTPUT_CHANNEL_LEFT=AUX0
AURA_ENGINE_OUTPUT_CHANNEL_RIGHT=AUX1
AURA_ENGINE_INPUT_DEVICE=Yamaha AG06MK2:capture
AURA_ENGINE_INPUT_CHANNEL_LEFT=AUX0
AURA_ENGINE_INPUT_CHANNEL_RIGHT=AUX1
```

Now that you have your device name in the `.env` file you can start *AURA Playout* with

```
$ docker compose up -d
```

2.7.3 Playout channel routing

Playout channels are routed this way:

Deploy Play-Out bundled with Recorder (optional)

By default the Docker-Compose will not deploy the the Aura-Recorder. If you want to deploy the Aura-Recorder on the same Host start the compose service with

```
$ docker compose --profile engine-recorder-enabled up -d
```

2.7.4 Configure the audio source locations

Engine Core is requires different audio sources in order to perform the playout.

Configure the location for fallback music

By default fallback audio is retrieved from the `fallback` folder. A local folder for any emergency playback, also called *Station Fallback*.

```
audio_fallback_folder="audio/fallback/"
```

All audio files inside are played in a randomized order, in situations where nothing is scheduled. The folder is being watched for changes. So you can add/remove audio on the fly.

This fallback feature is enabled by default, but can be turned off in via the configuration.

Instead of the fallback folder you can use a playlist in the `playlist` folder for fallback scenarios. Its default file name is `station-fallback-playlist.m3u` and located in:

```
audio_playlist_folder="audio/playlist"
```

Also this playlist is being watched for changes. You'll need to set the configuration option `fallback_type="playlist"` to enable this instead of the fallback folder.

Configure the audio source folder

This is the location for actually scheduled audio files. They are provided by Tank.

```
audio_source_folder="audio/source"
```

If you are running all AURA services on a single instance you should be fine with just creating a symbolic link to the relevant Tank folder (`ln -s $TANK_STORE_PATH $PLAYOUT_AUDIO_SOURCE`). But in some [distributed and redundant production scenario](#) you might think about more advanced options on how to sync your audio files between machines.

2.7.5 Features and how they work

Scheduler

Engine provide a scheduling functionality by polling external API endpoints frequently. Those API endpoints are provided by [Steering](#) to retrieve schedule information and [Tank](#) to retrieve playlist information. To define your schedule you'll also need [AURA Dashboard](#) which is an elegant web user interface to manage your shows, playlists and schedules.

Ideally any audio is scheduled some time before the actual, planned payout to avoid timing issues with buffering and preloading. Nonetheless, playlists can also be scheduled after a given calendar timeslot has started already. In such case the payout starts as soon it is preloaded.

If for some reason the payout is corrupted, stopped or too silent to make any sense, then this triggers a fallback using the silence detector (see chapter below).

Note: If you delete any existing timeslot in Dashboard/Steering this is only reflected in Engine until the start of the scheduling window. The scheduling window is defined by the start of the timeslot minus a configured offset in seconds (compare your Engine configuration).

Versatile playlists

It is possible to schedules playlists with music or pre-recorded shows stored on the **file system**, via external **streams** or live from an **line input** in the studio. All types of sources can be mixed in a single playlist.

The switching between types of audio source is handled automatically, with configured fadings applied.

Note: Any live sources or streams not specifying a length property, are automatically expanded to the left duration of the timeslot.

Default playlists

While a timeslot can have a specific playlist assigned, it is also possible to define default playlists for schedules and shows:

- **Default Schedule Playlist:** This playlist is defined on the level of some recurrence rules (*Schedule*). In case the timeslot doesn't have any specific playlist assigned, this playlist is broadcasted.
- **Default Show Playlist:** This playlist can be assigned to some show. If neither the specific timeslot playlist nor the default schedule playlist is specified the *default show playlist* is broadcasted.

If none of these playlists have been specified the *Auto DJ* feature of *Engine Core* takes over (optional).

Heartbeat Monitoring

Instead of checking all status properties, the Heartbeat only validates the vital ones required to run the engine. If all of those are valid, a network socket request is sent to a defined server. This heartbeat is sent continuously based on the configured `heartbeat_frequency`. The service receiving this heartbeat ticks can decide what to do with that information. One scenario could be switching to another Engine instance or any other custom failover scenario. Under `engine/contrib/heartbeat-monitor` you'll find some sample application digesting these heartbeat signals.

2.8 AURA Recorder

2.8.1 Deploy AURA Recorder

All required files can be found under `config/aura-recorder`. For the described commands change to that directory. Make sure to follow the [preparation steps](#) to add the user: `aura` first.

Then copy the `sample.env` to `.env` and set all values as needed.

```
$ cp sample.env .env
```

Update the configuration file

Update at least following environment variables in the `.env` file:

- `AURA_RECORDER_AUDIO_SOURCE`: The audio device from which the audio will be recorded. It is also possible to set a stream.
- `AURA_AUDIO_STORE`: The location where the recorder saves all recordings.

Make sure that the user `aura` can write to `AURA_AUDIO_STORE`.

```
$ source .env
$ mkdir -p "${AURA_AUDIO_STORE}/recordings/block"
$ chown -R aura:aura "${AURA_AUDIO_STORE}"
```

Using a ALSA audio device

Currently AURA Recorder only supports ALSA devices or audio streams. When using a audio interface, make sure to provide the sample format in the [override config](#). For future releases we plan to read the sample rate directly from the audio device.

Start the services with Docker Compose:

```
$ docker compose up -d
```

After successful start-up, you should see the `AURA_AUDIO_STORE` get populated.

2.9 Advanced Configuration

This chapter outlines ways to achieve custom and advanced setups.

2.9.1 Custom service configuration

In the previous steps you used the `.env` configuration files created in the Docker Compose directories `config/aura-web`, `config/aura-playout` and `config/aura-recorder`. In most common setup scenarios all configuration done with environment variables is sufficient.

For some more advanced setups or debug purposes, there are also sample configuration files for each service under `/opt/aura/config/services/sample-config` available. To overwrite any service configuration, simply copy its configuration file to the parent `services` folder.

Only use these overrides if you are an advanced user, or are advised to do so.

2.9.2 Nginx and Reverse Proxy

In the *AURA Web* setup all services are made reachable behind a reverse Proxy. This Nginx-based setup also supports SSL via Let's Encrypt.

Behind second Reverse Proxy on different Hosts

If you have another reverse proxy in your Organization, which terminates TLS and does the routing for all your other services, Aura-Web does support this by disabling Certbot (Let's Encrypt) and communicate via http. To do this you can just use port 80 of this Host. We assume what your hosts are communicating in your internal network, because all that communication is unencrypted http traffic.

Basic Config on 1st Reverse Proxy

- The 1st Reverse Proxy must listen on Port 443 (https) and needs to route your incoming aura-web traffic to your Aura-Web host to Port 80
- Your 1st RP needs to terminate TLS
- At the Moment we cannot support URL rewrites

Config Aura-Web on 2nd Host

- Aura-Web needs to listen on Port 80
- Disable Certbot

```
AURA_HOST_CERTBOT_ENABLE=false
```

- Keep `# AURA_HOST_PROTO=http` commented out!
 - Our internal components need to know that they are being served via https, for their redirects and external API endpoints to work.

1st and 2nd Reverse Proxy on same Host

At the moment running another reverse proxy on the same machine isn't officially supported with this setup, since there's no possibility to close ports with an override. Be aware that the Docker Compose by default opens ports 80 and 443. To keep this running you may need to adjust the nginx config and docker-compose.yml

Disable Nginx

If you wish to not use Nginx whatsoever, you can override the Docker Compose with the following content:

```
services:
  nginx:
    deploy:
      replicas: 0
```

This disables the internal Nginx. Keep in mind the nginx does a lot of heavy lifting to handle all redirections and URL rewrites correctly, which you will then need to handle manually.

If you wish to expand the nginx-config you can put further configs into the custom-folder. Keep in mind the configs need to end on .conf.

2.10 Update and Maintenance

This guide helps to update and monitor your AURA installation.

Docker commands can be executed from any location. Docker Compose commands are to be called from the docker-compose.yml folder.

To learn about additional commands and arguments, consult the [official Docker Compose command reference](#).

2.10.1 Upgrade to a new release

Update Containers

Before doing an update, check the [Release Notes](#) page, to learn what has been changed. For the case of a failed update, ensure you have *backups* of your data.

Follow these steps to update your services.

1. Switch to the new release

Pull the aura repository:

```
$ git pull
```

list available releases:

```
$ git tag
1.0.0-alpha1
1.0.0-alpha2
1.0.0-alpha3
```

and check out the new release:

```
$ git checkout tags/<release-version>
```

Replace `<release-version>` with any of the releases shown by `git tag`.

2. Update the configuration

Compare your `.env` file with the (potentially changed) `sample.env` and change your `.env` file accordingly. Take special note of new versions of the services. Carefully check the release notes for changes you might need to take into account. For `aura-playout`, you also want to compare the `sample.engine-core.ini` with your `engine-core.ini`.

3. Pull the new images from Docker Hub

```
$ docker compose pull
```

4. Re-create the containers

```
$ docker compose up -d
```

2.10.2 Logging and Monitoring

List running services

To get a list of services currently running use:

```
$ docker ps
```

Alternatively you can also run, giving additional details on the container composition:

```
$ docker compose ps
```

Logs

To continuously watch the logs of all services included in the current *Docker Compose*, type:

```
$ docker compose logs -f
```

To see the current logs of individual containers running with *Docker Compose*, execute:

```
$ docker compose logs -f --tail=100 <service-name>
```

This command prints the last hundred log lines, but also keeps watching for new entries.

If you prefer to access the logs directly, go to `logs`. Here you can find the log files for all services. Use `tail` to watch the log file:

```
$ tail -f -n100 <file>
```

The files in `logs` can be integrated to your favorite third-party monitoring software.

Prometheus Monitoring

In some [future release](#), we will provide a native [Prometheus Monitoring](#) integration.

2.10.3 Resetting, shutting down and cleaning up

Enforce creation of new containers

```
docker compose up -d --force-recreate
```

This recreates containers even if their configuration and image have not changed.

Stop and remove containers

If you wish to remove your deployment, all you need to do is shutting it down:

```
docker compose down
```

If you also wish to delete all data, including the Docker Volumes, you can run the following command:

```
docker compose down -v
```

Delete unused images

Since Docker does not automatically delete old images, they may pile up and occupy too much space. So you should delete them once in a while.

To delete images of containers not currently running, use:

```
docker system prune --all
```

This will not delete the Docker volumes. Volumes are the place, where the databases and the persistent data lives.

The `--all` option also removes unused images, not just dangling ones.

Delete all data

If you want a complete fresh installation, you may need to delete all data, avoiding any conflicts.

Be aware this deletes all volume data, including the ones not part of the AURA installation.

```
docker system prune --all --volumes
```

2.10.4 Backup Strategy

Include following filesystem locations to your backup system. Some of these are only available in certain deployment bundles, such as `aura-web`, `aura-playout` or `aura-recorder`.

Filesystem location	Deployment bundle	Description
<code>aura/config</code>	all	All custom and sample configuration files
<code>aura/log</code>	all	The reported log files
<code>aura/audio</code>	all	The audio store, holding all audio and playlists
<code>/var/lib/docker/volumes</code>	all	All Docker Volumes

Every Docker Volume is prefixed with the specific deployment bundle. Below you find commands for backing up all Docker Volumes.

For Database specific backups you can use the description from [Upgrade the database version](#)

Backup & Restore Docker Volumes

To backup all AURA related Docker Volumes, commands for backup and restore are provided.

To create a backup run:

```
sudo make aura-volume.backup BACKUP_PATH=/your/backup/path
```

This will create a `tar.gz` archive for each volume added with a timestamp, to your desired path.

To restore a volume backup run:

```
sudo make aura-volume.backup BACKUP_FILE=/your/backup.tar.gz VOLUME_NAME=aura-steering
```

Avoid data corruption, ensure volume is not in use

Before running the restore procedure, make sure the specific volume is not in use, otherwise you may end up with corrupted data!

2.10.5 Special Operations

Log into a container

To *bash* into an already running container execute:

```
docker compose exec -it steering bash
```

To log into the database of a PostgreSQL container you can run:

```
docker compose exec steering-postgres psql -U steering
```

Manage OpenID Clients

Open ID clients are used for authorized access to *Steering*.

For example *Dashboard* and *Tank* need to be registered as clients, in order to communicate with *Steering*.

Usually they are created during the initialization step of *Aura Web*. In some case you may want to update them, to fix an broken installation. Or you might want to register any 3rd party service at *Steering*.

Create OpenID Connect clients

To create an OpenID Connect client for Dashboard, update `AURA_DASHBOARD_OIDC_CLIENT_ID` in your `.env` and run:

```
$ docker compose run --rm steering create_oidc_client.dashboard
```

To create an OpenID Connect client for Tank, update `AURA_TANK_OIDC_CLIENT_ID` in your `.env` and run:

```
$ docker compose run --rm steering create_oidc_client.tank
```

These commands will fail if the `client_id` is already existing in Steering. In that case you need to delete the existing client first.

Delete OpenID Connect clients

If you need to replace an existing `client_id`, you can delete them in Steering's administration interface at "*Steering > OpenID Connect Provider > Clients*".

If you cannot access Steering's administration interface, you can delete the clients by running:

```
$ docker compose run --rm steering delete_oidc_clients
```

This will delete the existing OpenID Connect clients for Dashboard and Tank.

You can then re-create as described *above*.

Renew schedules for a new year

By default, `steering` generates timeslots from the first date of a schedule, until either the last date of the schedule, the end of the year or defined number of days after the first date.

To renew the schedules for a new year you can run the `addtimeslots` management command, specifying the year you want to add timeslots for.

For example, for the year 2024 you execute:

```
$ docker compose exec steering poetry run ./manage.py addtimeslots 2024
```

This will add timeslots to the schedules that don't have a "once" recurrence and where the last date is not set.

You can add the `--dry-run` option to see the results, without actually updating anything.

```
$ docker compose exec steering poetry run ./manage.py addtimeslots --dry-run 2024
```

Upgrade the database version

The postgres-version is saved in the POSTGRES_VERSION variable. When upgrading Postgres, it is not sufficient to change this version though. New major versions of Postgres cannot read the databases created by older major versions. The data has to be exported from a running instance of the old version and imported by the new version.

Thankfully, there is a Docker container available to automate this process. You can use the following snippet to upgrade your database in the volume aura-web_steering_db_data, keeping a backup of the old version in aura-web_steering_db_data_old:

```
# Replace "9.4" and "11" with the versions you are migrating between.
export OLD_POSTGRES=9.4
export NEW_POSTGRES=11
docker-compose stop steering-postgres
docker volume create aura-web_steering_db_data_new
docker run --rm \
  -v aura-web_steering_db_data:/var/lib/postgresql/${OLD_POSTGRES}/data \
  -v aura-web_steering_db_data_new:/var/lib/postgresql/${NEW_POSTGRES}/data \
  tianon/postgres-upgrade:${OLD_POSTGRES}-to-${NEW_POSTGRES}
# Add back the access control rule that doesn't survive the upgrade
docker run --rm -it -v aura-web_steering_db_data_new:/data alpine ash -c "echo 'host all_
↳all all trust' | tee -a /data/pg_hba.conf"
# Swap over to the new database
docker volume create aura-web_steering_db_data_old
docker run --rm -it -v aura-web_steering_db_data:/from -v aura-web_steering_db_data_old:/
↳to alpine ash -c "cd /from ; mv . /to"
docker run --rm -it -v aura-web_steering_db_data_new:/from -v aura-web_steering_db_data:/
↳to alpine ash -c "cd /from ; mv . /to"
docker volume rm aura-web_steering_db_data_new
```

Please double check all values and that your local setup matches this. Of course this needs to be done for all postgres-dbs.

Automating Image Updates

You may want to have a testing instance, where you automatically update to the latest images.

In that case [Watchtower](#) can be used to keep the images up to date.

By adding the Watchtower service to your compose file will poll for new images every 60 seconds:

```
---
services:
  watchtower:
    command: --interval 60
    container_name: watchtower
    image: containrrr/watchtower:1.5.3
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
```

Watchtower will stop and restart outdated containers automatically.

There is a sample [Docker Compose](#) file including Watchtower.

Overriding the docker-compose.yml

If you need to make changes to the `docker-compose.yml` you can create a `docker-compose.override.yml` and make the necessary adjustments in there. That way, your adjustments won't create conflicts.

Deploy other Docker Images

If you prefer some individual deployment scenario, you can also run single Docker Images.

These Docker images are hosted on <https://hub.docker.com/u/autoradio>.

Work in progress

These images are not yet fully documented. We will update the documentation on Docker Hub as we move along. If you need such image please consult the documentation and *Makefiles* in the relevant repositories.

2.11 Frequently asked questions (FAQ)

2.11.1 General

When running any Docker Compose bundle, I get permission errors in various places. What am I doing wrong?

You probably missed certain instructions of the installation steps. Carefully re-read the installation guide and check each and every step. There are certain initialization steps, which create the required directories or set according permissions. It is also possible that you are starting services with the wrong user. For example you are running Docker as `root` or you are having wrong ownership of the files and directories in the `/opt/aura` directory.

After changing some setting in the configuration file, why does it have no effect?

Double-check if the configuration line is commented out. Remove the `#` at the beginning of the line, to enable the setting. Any commented out settings, do not necessarily indicate the default setting. Also, check the descriptions provided with the settings and compare with instructions in the documentation.

When logging out from the server via SSH, the Docker Containers stop running. Why is that?

It looks like your Docker Engine installation is bound to your session. Install Docker running as a daemon. We also recommend running Docker Engine in the *rootless* using *systemd*.

Why is there a blank page or 404, when browsing to the Dashboard or after clicking login on my fresh installation?

This can have multiple reasons:

- If you have installed AURA locally remember not to use `localhost` or `127.0.0.1`. This is the most common pitfall.
- You have not initialized the OIDC clients correctly.
- Your containers are not up- and running, or they are not healthy. Check the running containers and logs.

For all of these scenarios, check the installation steps for details.

2.11.2 Playout (Pipewire)

Why is my audio and any timeslot transitions delayed for 15-20 seconds during playout?

Most likely you have *PulseAudio* installed or running your playout through the *Pipewire PulseAudio* plugin. Get rid of *PulseAudio* completely and retry.

2.11.3 Playout (ALSA)

I am using the default audio device. How can I set another default device?

You can check the systems default audio hardware by executing `aplay -L` on the command line.

You can set the default device in `/etc/asound.conf` or `~/asoundrc`.

How can I retrieve available ALSA audio devices?

- To see only the physically available sound cards: `cat /proc/asound/cards`
- To see sound cards with all additional devices (e.g. HDMI): `aplay -l`
- To see devices configured by ALSA with additional plugins: `aplay -L`
- The default devices that should be used: `aplay -L | grep default`

I have configured an audio device, but why do I still hear no sound (bare metal installation)?

To test if you device is able to output audio at all, independently from Engine Core, try executing `speaker-test`. Also checkout out the `-D` argument to test specific devices. If you system doesn't provide `speaker-test` you have to install or use your preferred way of testing also audio.

I have configured an audio device but still hear no sound (Docker installation)

If you are running Engine Core using Docker, run the aforementioned `speaker-test` from within your docker container by perform following:

1. Bash into the container using `docker exec -it aura-engine-core bash`
2. Now run `speaker-test`. If that's working, you now know that your audio device is at least available from within Docker and you'll need to further check your Liquidsoap device configuration.
3. Next you can run `liquidsoap tests/test_alsa_default.liq`. This is a basic script which tries to play the supplied MP3 using the default ALSA device.

Why I am getting an `clock.wallclock_alsa:2 ... error?`

The error message also displays `Error when starting output lineout: Failure("Error while setting open_pcm: No such file or directory")`.

Assure you have set the correct ALSA device ID. Review the audio interface configuration settings and verify if the default settings in the configuration file for your input and output devices are valid device IDs.

Also verify if the executing user (e.g. `aura`) belongs to the group `audio`.

How to solve Error when starting `output_lineout_0: ...?`

The error message continues as `Failure("Error while setting open_pcm: Device or resource busy")!`.

You probably have set a wrong or occupied device ID. The device could be already reserved by another software using the ALSA sound system. Or you might be accessing a device using ALSA which is already assigned to the *PulseAudio* sound system. Here it could help to [remove the device from PulseAudio](#) before accessing it.

How to avoid stutter, hangs, artifacts or in general glitchy sound?

This can have various reasons, but first of all it's good to check the `engine-core.log` file. Also check your CPU usage. Lastly review the settings of your audio device.

Incorrect ALSA buffer settings: If the ALSA settings provided by your system are not working cleanly the Engine Core settings provide to option to override parameters such as `alsa_buffer`. The correct settings are individual to the used soundcard but in general this is a tricky topic and deeper ALSA knowledge is very helpful.

These problems occur while having Icecast streaming enabled: Try to reduce the quality of the stream, especially when you are experiencing hangs on the stream. Check your Icecast connection. Is it up and running? Maybe there is some authentication issue or an *Icecast limitation for max clients*.

The hardware is hitting its limits: Also check the relevant logs and the system utilization. Are there other processes using up the machines resources? You might even be hitting the performance limit of your hardware. Maybe using a realtime linux kernel could help too.

2.12 Resources

Nothing to see here. Yet.

DEVELOPER GUIDE

This guide holds general information for AURA architecture and development.

3.1 Architecture

Find details on the AURA Architecture [here](#).

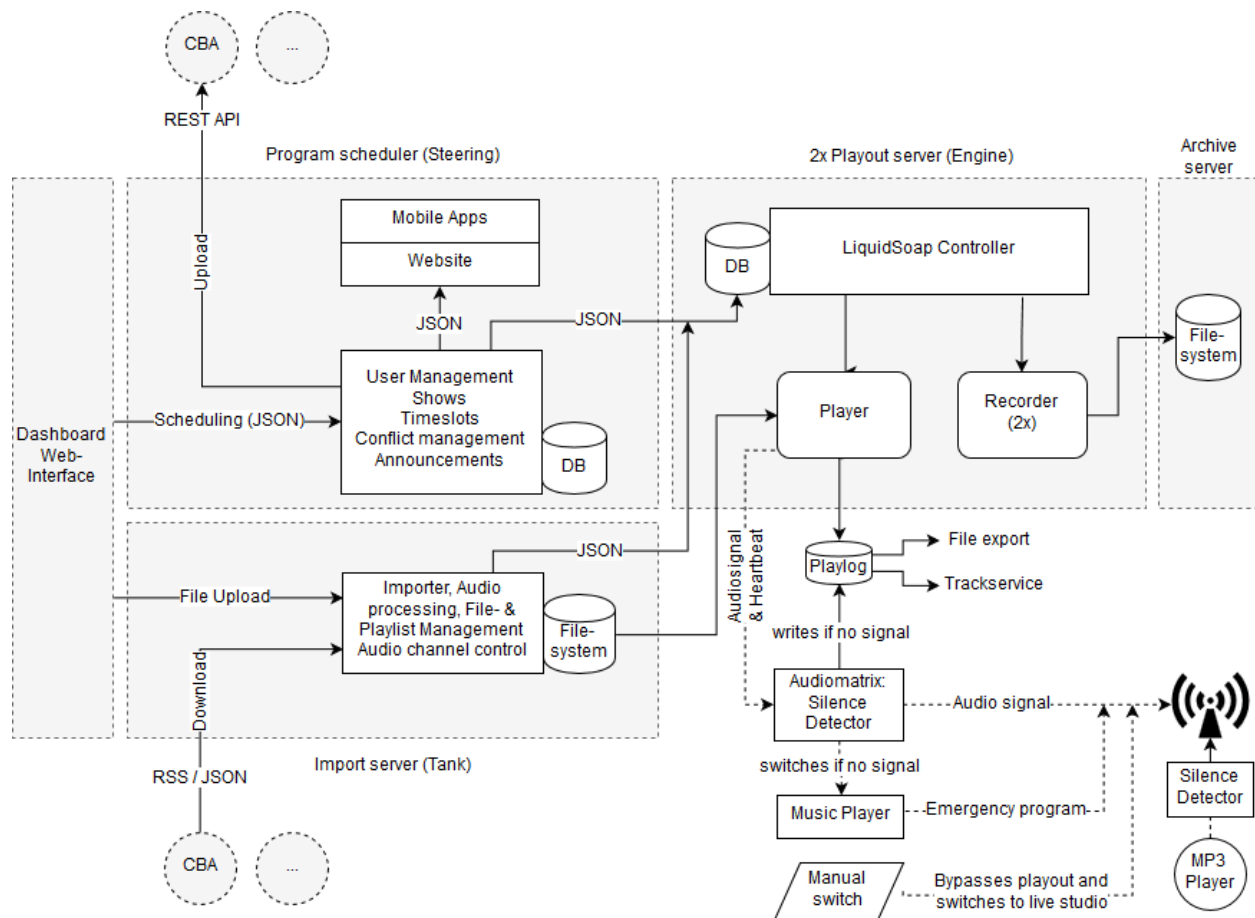
3.1.1 Architectural principals

Some of our core organisational and architectural requirements for AURA are:

- **modular architecture:** the whole suite should be made up of modular components which could be exchanged with other custom components
- **transparent API:** every component shall provide a well-documented API through which other components can interact with it, ideally as a REST API.
- **reuse of existing components:** we do not want to reinvent the wheel. Several stations already developed single components as free software and we can adapt and build on those
- **modern frameworks:** we do not code from scratch but use modern application development frameworks which provide maintainability as well as security

Outdated diagram

The following component diagram doesn't reflect all the details of the current implementation. It will be updated at some point.



3.1.2 Diagrams

Network Diagram

Check out the provided *Deployment Scenarios* on ideas how the individual projects can be integrated within your infrastructure.

Data Model

Simplified Data Model

This data model is an abstraction of the main entities used by Steering and Tank.

Steering Data Model

This is the Steering data model as of March 2024. Parts of Django and other applications that are not directly involved were omitted.

Tank Data Model

This is the Tank data model as of March 2024.

3.1.3 Conflict Resolution for the scheduling timetable

Check out the [Conflict Resolution Documentation](#) page.

3.2 Development

3.2.1 Coding Conventions and Guidelines

Here you find an overview of our conventions on coding and version control.

Also have a look at our [Contribution Guidelines](#).

git

- We use [GitHub Flow](#)
- Keep the `main` branch stable, as releases are derived from it. When contributing a breaking change, like API changes, inform the team about the intended merge.
- When contributing code, create a *Merge Request*, and assign the repository maintainer for review. Do not *squash commits* when performing the merge.
- We write changelogs in the [keep a changelog](#) format. Remind yourself to update the `CHANGELOG.md` before committing code. Remember that changelogs are meant for humans and should reflect the end-user value. When *doing a release* we use the changelog as our *Release Notes*. When providing code via a *Merge Request* the maintainer will remind you to update the changelog too :-)
- Avoid crunching commits and rebasing; set `git config pull.rebase false` to use *recursive* as your default merge strategy
- We use *conventional commits*
- Mention related ticket IDs where applicable, like `#123` or `play#123` when cross-referencing between repositories. For example you can close tickets by using `closes #123`.
- Use [atomic commits](#)

Code

Python

We use [Black](#) with default settings, enforced by [Flake8](#).

We use the default settings, except for a maximal line-length of 99 characters. If you are using a Black IDE integration, think about adapting its settings.

For code documentation we use the [Flake8 Docstrings](#) extension with [Google-style formatting](#) enabled (`docstring-convention=google`).

ECMAScript and TypeScript

We use [ESLint](#) as the common Linter. When your code is based on [VueJS](#) or [Svelte](#), install the official IDE extensions. These extensions provide additional style checks and auto-formatting options.

API

We utilize an *API-first* approach. APIs are specified using OpenAPI 3. Find the API at api.aura.radio.

All the main aspects are documented within the spec. In some cases you may need some additional documentation in the docs. For example the [API: Schedules and conflict resolution](#) document can be found in “*Developer Guide -> Misc*”.

API-first

At the moment only [Engine API](#) is based on API-first. [Steering API](#) and [Tank API](#) are momentarily generated out of the code base.

Testing

When providing new features or refactoring code, please provide test cases. We do not expect 100% test coverage but are aiming for having the most important usage scenarios covered by automated CI/CD test suites.

Test and Demo Instances

We have two instances which are automatically and transparently deployed using GitLab:

- [dashboard.aura.radio](#) - Developer/Testing instance holding the latest codebase of the `main` branches.
- [demo.aura.radio](#) - Demo instance with a stable release, based on a release tag configured in GitLab.

dashboard.aura.radio

We use the [gitlab-ci](#) for Deploying specific versions of aura.

- You Can adjust the Container versions at [gitlab-ci](#)

```
AURA_STEERING_VERSION: "main"
AURA_TANK_VERSION: "main"
AURA_DASHBOARD_VERSION: "main"
AURA_DASHBOARD_CLOCK_VERSION: "main"
```

- you can choose between Docker Registry or Gitlab-Registry at [gitlab-ci](#)
 - DOCKER_REGISTRY: "docker" for Docker hub
 - DOCKER_REGISTRY: "gitlab" for Gitlab registry

After committing your desired changes, Gitlab auto deploy this version. Note that the Pipeline will only run on protected branches.

demo.aura.radio

When a new Tagged Release is published, Gitlab will deploy this release.

There is a manual apply before deploy to ensure all Container Images with this Tag are published. Apply then Docker Images are released!

After run apply gitlab opens a popup, where you insert Variables. There is no need for changing the Vars.

Just run `Run job`

Auto Deployment based on feature branches

While this is currently not used, the Auto Deployment feature also provides custom builds based on feature branches.

For all components, feature branches automatically build Docker containers and push them to the [gitlab-registry](#).

To roll out new features, the [Compose-File](#), must be set to the corresponding tag of the feature branch.

After the commit, the version will be rolled out automatically.

Documentation

The general documentation is located in `aura/docs` and hosted at [docs.aura.radio](#). When working on any component, also check if this documentation has to be updated or extended.

3.2.2 Developer Installation

For development, we recommend the native installation as outlined in the [README](#) of individual [repositories](#).

Docker Compose Deployments

For production we highly recommend to run AURA using Docker and Docker Compose as outlined in the [Administration Guide](#). But also as a developer you can benefit from the ease of an Docker Compose installation. For example when developing some Engine feature, you may want to run AURA Web with Docker Compose while testing.

Prepare your Development Environment

It is recommended to clone all projects for example in such folder structure:

```
~code/aura/aura
~code/aura/steering
~code/aura/dashboard
~code/aura/engine
~code/aura/engine-api
~code/aura/engine-core
...
```

Order of configuration

After that, you need to configure the projects in following order:

Web Projects

1. [Steering](#) - Administration interface for schedules and programme information.
2. [Dashboard](#) - Frontend to manage schedules, program info and audio files.
3. [Tank](#) - Upload, pre-processing and storage of the audio files.
4. ...

Play-out Projects

1. [Engine Core](#) - Playout-engine to deliver the actual radio to the audience.
2. [Engine API](#) - API Server to provide playlogs and information for the studio clock.
3. [Engine](#) - Scheduling and remote control for the playout-engine.
4. ...

Configuring the OpenID Clients

Dashboard and Tank authenticate against Steering using OpenID. We use [OpenID Connect \(OIDC\)](#) to implement OpenID.

Check out the [OIDC configuration](#) page, on how to get them talk to each other.

3.2.3 Component Documentation

For more detailed documentation read the `README` files in the individual repositories.

You get an overview of all repositories at code.aura.radio.

3.3 Release Management

3.3.1 Semantic Versioning

Release names are defined according to the [SemVer 2.0.0](#) versioning scheme.

Semantic versioning of releases in CI/CD Pipelines

The chosen git tag will be the release name and the version of the Docker Image. Therefore the pipeline script will enforce it to conform to [Semantic Versioning 2.0.0](#).

3.3.2 Keep a Changelog

For changelogs we use the format suggested by [Keep a Changelog](#). Try to follow best practices when writing the changelog. But remember, changelogs are for humans. So do apply best practices when compiling your changelog.

The general AURA changelog is located in `CHANGELOG.md`.

Additionally, all individual service repositories hold their own changelog file. At each release they are bundled in the `aura/changelog` directory.

When writing changelogs think about to

- use **BREAKING:** prefixes to indicate breaking changes.
- avoid multi-line statements which include line-breaks. This breaks the release script.
- clarity over quantity, because changelogs are for humans.

Changelogs are for humans!

Avoid any technical details on internals. But documenting API or CLI changes are okay. Try to read it from a end-user perspective. Learn more at keepachangelog.com.

3.3.3 Current Release

You can find available releases at releases.aura.radio or by executing `git tag`.

Find releases of individual services in the [Gitlab](#) repository page under *Deployments > Releases*.

3.3.4 Release Workflow

The release workflow at a glance:

1. Cycle Planning in the beginning of the release cycle:
 1. Define the date for code-freeze e.g. Monday evening of the the 6th week.
 2. Define the release testers e.g. 1-2 people are sufficient.
 3. Define the release date e.g. Thursday of the 6th week. The cooldown phase can be used, when there are huge, unexpected issues.
2. When code-freeze hit, repository maintainers *release their services*. When done, they inform the release manager.
3. The release manager starts *preparing the release*. When done, the release testers are informed.
4. The release testers start testing, mainly if the suite can be still installed according to the documentation. When issues appear, relevant tickets are re-opened and developers are informed. If successful the product owner and release manager are informed to proceed with the release.
5. The release manager *performs the release*.

Find the detailed steps per role below.

Releasing a service

This release step is **done by the repository maintainers**.

To perform a service release do:

1. While you should keep the `CHANGELOG.md` up to date at all times, now is a good time to check if you have forgotten something. Also, review if it fits the *common format*.
2. In the changelog, update the version from `[Unreleased]` to the current version to be released and the current date. Remove all the unused sections. Commit and push.
3. Bump the version in the file holding the current version. Commit and push the version update.
4. Release the version by running `make release`. This command tags and pushes the current branch.
5. Then create a new template section with the `[Unreleased]` header and all required sections.

As soon a version tag is pushed, the *CI/CD Pipeline* performs these steps:

- Create a GitLab release with the provided `CHANGELOG.md` as its release notes.
- Pushes the image to [Docker Hub](#).
- Build the *Docker Image* and automatically tags the release with `latest` and `<release version>`.

Releasing the AURA software bundle

The release of a complete software bundle is triggered from within the aura repository and **performed by the release manager**.

Before proceeding review the main AURA changelog in `aura/CHANGELOG.md`.

Preparing the release

1. Create a new branch for the release: `git checkout -b %VERSION%-prep`. Note the `-prep` suffix for release preparation, as the actual `%VERSION%` branch needs to stay unused for the actual release tag branch.
2. In `aura/CHANGELOG.md` change the version identifier `[Unreleased]` to the release version and add the current date.
3. Bump the version of the aura repository with `poetry version <version>`.
4. Update the versions of referenced services in the *Docker Compose* `config/<docker compose>/sample.env` files and in `.env.version`.
5. Update the configuration files, fixtures and changelogs by running

```
make dev.prepare-release
```

This command pulls all configuration files, fixtures and changelogs from the individual repos identified by the versions in `.env.version`. The changelogs are stored `.tmp` and are merged into a file named `merged-CHANGELOG-%AURA_VERSION%.md`. Commit any changed configs and fixtures.

6. Commit and push the branch.

Now inform the testers to check-out and test the given branch.

Performing the release

After the **testers inform the product owner and development team**, that the *release criteria* is met, we can proceed.

In case developers have done any last minute bugfixes and re-released their services, ensure all the configuration and changelog files are still up-to-date by executing

```
make dev.prepare-release
```

once more.

1. Take the contents of `.tmp/merged-CHANGELOG-%AURA_VERSION%.md` and paste it into a new section in `docs/release-notes.md`.
2. Now we have to start thinking: What should be in the header of the release notes? Comprehend what this release delivers by checking the roadmap and contents of the changelog below.
3. Review and commit the changes.
4. To ship the release, run

```
make dev.release
```

5. Check out `main` and merge everything changed, except the version updates. They should remain as `main` for all services. Alternatively cherry-pick the relevant commits or update the relevant version strings to `main` after the merge.

6. Update `aura/CHANGELOG.md` with a new template section for the next release.
7. Push the changes to `main`.
8. Delete the branch used for release preparation.
9. Inform the product owner and team that the release is done.

3.4 Misc

3.4.1 OpenID Client Configuration

AURA is using [OpenID](#) for authentication and authorizing access to restricted API endpoints.

More specifically we are using [OpenID Connect \(OIDC\)](#) for the OpenID handshakes.

[Steering](#) is the central OpenID provider. All applications requesting access, need to get an authorization from Steering.

Those applications are called *OIDC clients*.

Required OIDC Clients

In order to properly setup AURA, you'll need to configure OpenID clients for Dashboard and Tank.

The registration and configuration steps below use *the default hosts & ports*.

In case of a [Production Deployment](#) you'll probably have substitutions like following:

- Steering: `localhost:8080` → `aura-host.org/admin`
- Dashboard: `localhost:8000` → `aura-host.org`
- Tank: `localhost:8040` → `aura-host.org/tank`

Registering clients at Steering

Registering OIDC clients on the command-line

First navigate to your Steering project location.

1. Create an RSA Key

```
$ poetry run ./manage.py creatorsakey
```

2. Create OIDC client for Dashboard

```
$ poetry run ./manage.py create_oidc_client dashboard public -r "id_token token" -u https://localhost:8080/oidc_callback.html -u https://localhost:8080/oidc_callback_silentRenew.html -p https://localhost:8080/
```

Important: Remember to note the client id and secret for the configuration section below.

1. Create OIDC client for Tank

```
$ poetry run ./manage.py create_oidc_client tank confidential -r "code" -u https://localhost:8040/auth/oidc/callback
```

Important: Remember to note the client id and secret for the configuration section below.

Registering OIDC clients via the admin interface

Follow these three steps to register Dashboard and Tank in the OpenID admin section of Steering.

Create an RSA Key

In the admin interface navigate to *OpenID Connect Provider* and *generate a RSA Key*.

Create OIDC client for Dashboard

Here you'll need to choose following settings:

```
Client Type: Public
Response Type: id_token token (Implicit Flow)
JWT Algorithm: RS256
Require Consent?: No
Reuse Consent?: Yes
```

And enter these redirect URLs:

```
http://localhost:8080/static/oidc_callback.html
http://localhost:8080/static/oidc_callback_silentRenew.html
```

Note, that these URLs have to match exactly the ones you configure in your `.env.development` or `.env.production` files here in the dashboard source. This also means that if you use `localhost` in steering, you must not put `127.0.0.1` or any equivalent in your dashboard config, but use exactly the same string (and vice versa).

Note the *Client ID* to use in your Dashboard config file.

TODO

Replace image with a current screenshot of Steering

Create OIDC client for Tank

Here you'll need to choose following settings:

```
Client Type: Confidential
Response Type: code (Authorization Code Flow)
JWT Algorithm: RS256
Require Consent?: No
Reuse Consent?: Yes
```

And enter that redirect URL:

```
http://localhost:8040/auth/oidc/callback
```

Note the *Client ID* and secret to use in your Tank config file.

TODO

Replace image with a current screenshot of Steering

Setting the client configuration

When configuring a client, always remind yourself to use the actual hostname. When using the IP address for OIDC redirect URLs you might get unexpected behaviour or being unable to authenticate at all.

Configuring Dashboard

In the Dashboard folder, edit your `.env.production` or `.env.development` respectively, and carefully review if these URLs are matching the ones in the the Steering client settings. These URLs should match your Dashboard host:

```
VUE_APP_API_STEERING_OIDC_REDIRECT_URI = http://localhost:8080/oidc_callback.html
VUE_APP_API_STEERING_OIDC_REDIRECT_URI_SILENT = http://localhost:8080/oidc_callback_
↪silentRenew.html
```

Then set the client id and secret, which you noted from the previous step:

```
VUE_APP_OIDC_CLIENT_ID = %YOUR_ID%
```

Additionally, confirm that your configured Steering URL and port is also matching the instance Steering is running at:

```
VUE_APP_API_STEERING_OIDC_URI = http://localhost:8000/openid
```

Configuring Tank

In the Tank configuration file `tank.yaml` replace `${OIDC_CLIENT_ID}` and `${OIDC_CLIENT_SECRET}` with your client ID and secret, or set the environment variables accordingly.

Also review the given URLs.

```
oidc:
  issuer-url: http://localhost:8000/openid
  client-id: ${OIDC_CLIENT_ID}
  client-secret: ${OIDC_CLIENT_SECRET}
  callback-url: http://localhost:8040/auth/oidc/callback
```

3.4.2 Recurrence rules

The following recurrence rules are supported by default if the `rrule.json` fixture is loaded.

name	freq	interval	by_set_pos	count	by_weekdays	notes
once	0			1		1
daily	3					

continues on next page

Table 1 – continued from previous page

name	freq	interval	by_set_pos	count	by_weekdays	notes
weekly	2	1				2
business days	2	1			0,1,2,3,4	
weekends	2	1			5,6	
bi-weekly	2	2				Page 70, 2
four-weekly	2	4				Page 70, 2
monthly on the first	1	1	1			3
monthly on the second	1	1	2			4
monthly on the third	1	1	3			5
monthly on the fourth	1	1	4			6
monthly on the fifth	1	1	5			7
monthly on the last	1	1	-1			8
bi-monthly on the first	1	2	1			Page 70, 3
bi-monthly on the second	1	2	2			Page 70, 4
bi-monthly on the third	1	2	3			Page 70, 5
bi-monthly on the fourth	1	2	4			Page 70, 6
bi-monthly on the fifth	1	2	5			Page 70, 7
bi-monthly on the last	1	2	-1			Page 70, 8
three-monthly on the first	1	3	1			Page 70, 3
three-monthly on the second	1	3	2			Page 70, 4
three-monthly on the third	1	3	3			Page 70, 5
three-monthly on the fourth	1	3	4			Page 70, 6
three-monthly on the fifth	1	3	5			Page 70, 7

continues on next page

Table 1 – continued from previous page

name	freq	interval	by_set_pos	count	by_weekdays	notes
three-monthly on the last	1	3	-1			Page 70, 8
four-monthly on the first	1	4	1			3
four-monthly on the second	1	4	2			4
four-monthly on the third	1	4	3			5
four-monthly on the fourth	1	4	4			6
four-monthly on the fifth	1	4	5			7
four-monthly on the last	1	4	-1			8

The actual IDs are not *hard-coded* in any way in **steering**, only the parameters of the recurrence rules are used to create the repetitions.

The only requirement is that the recurrence rule ID in your API call, or the dashboard, matches the recurrence rules store in the **steering** database.

If the frequency is either weekly or monthly, the weekday of the first date in the schedule determines the weekday of the repetition.

¹ The parameter `freq=0` actually describes a yearly frequency, but together with `count=1`, it achieves what we want: A single timeslot on the first date of the schedule.

² If the first date of the schedule is “2023-01-02” (a Monday), the resulting recurrence is weekly every Monday if `interval=1`, biweekly on Monday if `interval=2` or four-weekly on Monday if `interval=4`.

³ If the first date of the schedule is “2023-01-02” (a Monday), the resulting recurrence is monthly on the first Monday if `interval=1`, bi-monthly on the first Monday if `interval=2`, three-monthly on the first Monday if `interval=3`, and four-monthly on the first Monday if `interval=4`.

⁴ If the first date of the schedule is “2023-01-02” (a Monday), the resulting recurrence is monthly on the second Monday if `interval=1`, bi-monthly on the second Monday if `interval=2`, three-monthly on the second Monday if `interval=3`, and four-monthly on the second Monday if `interval=4`.

⁵ If the first date of the schedule is “2023-01-02” (a Monday), the resulting recurrence is monthly on the third Monday if `interval=1`, bi-monthly on the third Monday if `interval=2`, three-monthly on the third Monday if `interval=3`, and four-monthly on the third Monday if `interval=4`.

⁶ If the first date of the schedule is “2023-01-02” (a Monday), the resulting recurrence is monthly on the fourth Monday if `interval=1`, bi-monthly on the fourth Monday if `interval=2`, three-monthly on the fourth Monday if `interval=3`, and four-monthly on the fourth Monday if `interval=4`.

⁷ If the first date of the schedule is “2023-01-02” (a Monday), the resulting recurrence is monthly on the fifth Monday if `interval=1`, bi-monthly on the fifth Monday if `interval=2`, three-monthly on the fifth Monday if `interval=3`, and four-monthly on the fifth Monday if `interval=4`.

⁸ If the first date of the schedule is “2023-01-02” (a Monday), the resulting recurrence is monthly on the last Monday if `interval=1`, bi-monthly on the last Monday if `interval=2`, three-monthly on the last Monday if `interval=3`, and four-monthly on the last Monday if `interval=4`.

3.4.3 API: Schedules and conflict resolution

This document outlines handling conflict resolution using the API.

Find an overview of the API spec at api.aura.radio.

To learn about conflict resolution check out *Timeslot Collision Detection* in the User Guide.

Overview

Creating timeslots is only possible by creating/updating a schedule.

When creating/updating a schedule by giving the schedule data:

1. Projected timeslots are matched against existing timeslots
2. API returns an array of objects containing 2.1. the schedule's data 2.2. projected timeslots (array of objects), including an array of found collisions (objects) and possible solutions (array)
3. To resolve, POST/PUT the "schedule" object and the "solutions" objects to `/api/v1/shows/{show_pk}/schedules/{schedule_pk}/`

Create/update schedule

- To create: POST `/api/v1/shows/{show_pk}/schedules/`
- To update: PUT `/api/v1/shows/{show_pk}/schedules/{schedule_pk}/`

To start the conflict resolution POST/PUT the schedule object with key "schedule" containing:

Variable	Type	Meaning
byWeek-day ¹	int	Weekday number: Mon = 0, Sun = 6
rrule ^{1, 2}	int	Recurrence rule
firstDate ¹	date	First date of schedule (e.g. "2017-01-01")
startTime ¹	time	Start time of schedule (e.g. "16:00:00")
endTime ¹	time	End time of schedule (e.g. "17:00:00")
lastDate ³	date	Last date of schedule (e.g. "2017-12-31")
isRepetition	boole	Whether the schedule is a repetition (default: false)
default-PlaylistId	int	A tank ID in case the timeslot's playlistId is empty: What is aired if a single timeslot has no content? (default: null)
show ¹	int	Show the schedule belongs to
addDaysNo	int	Add a number of days to the generated dates. This can be useful for repetitions, like "On the following day" (default: null)
addBusiness-DaysOnly	boole	Whether to add <i>addDaysNo</i> but skipping the weekends. E.g. if weekday is Friday, the date returned will be the next Monday (default: false)
dryrun	boole	Whether to simulate the database changes. If true, no database changes will occur, instead a summary is returned of what <i>would</i> happen if dryrun was false. (default: false)

¹ These parameters are required.

² The full set of recurrence rules supported by default is loaded from the `rrule.json` fixture.

³ If `lastDate` is not provided, timeslots will be generated until the end of the year if `AUTO_SET_LAST_DATE_TO_END_OF_YEAR` is `True`, otherwise timeslots will be generated until `AUTO_SET_LAST_DATE_TO_DAYS_IN_FUTURE` after the `startDate` and `endDate` will remain unset.

Return

After sending the schedule's data, the response will be in the form of:

```
{
  /* Projected timeslots to create. Array may contain multiple objects */
  "projected": [
    {
      "hash": "2018011614300020180116160000041",
      "start": "2018-01-16 14:30:00",
      "end": "2018-01-16 16:00:00",
      /* Collisions found to the projected timeslot. Array may contain multiple
      ↪ objects */
      "collisions": [
        {
          "id": 607,
          "start": "2018-01-16 14:00:00",
          "end": "2018-01-16 15:00:00",
          "playlistId": null,
          "show": 2,
          "showName": "FROzine",
          "isRepetition": false,
          "schedule": 42,
          "memo": "",
          "noteId": 1 /* A note assigned to the timeslot. May not exist */
        }
      ],
      "error": "An error message if something went wrong. If not existing or empty,
      ↪ there's no problem",
      /* Possible solutions to solve the conflict */
      "solutionChoices": [
        "ours-start",
        "theirs",
        "ours",
        "theirs-start"
      ]
    },
    "solutions": {
      /* Manually chosen solutions by the user (if there's a key it has to have a
      ↪ value):
      Key is the hash of the projected timeslot while value must be one of
      ↪ 'solutionChoices' */
      "2018011614300020180116160000041": ""
    },
    "notes": {
      /* To reassign an existing note to a projected timeslot, give its hash as key
      ↪ and the note id as value (may not exist) */
      "2018011614300020180116160000041": 1
    },
    "playlists": {
      /* To reassign playlists to a projected timeslot, give its hash as key and the
      ↪ playlist id as value (may not exist) */
      "2018011614300020180116160000041": 1
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

},
/* The schedule's data is mandatory for each POST/PUT */
"schedule": {
  "rrule": 4,
  "byWeekday": 1,
  "show": 3,
  "firstDate": "2018-01-16",
  "startTime": "14:30:00",
  "endTime": "16:00:00",
  "lastDate": "2018-06-28",
  "isRepetition": false,
  "defaultPlaylistId": null,
  "automationId": null,
  "dryrun": false
}
}

```

Solutions

The "solutionChoices" array contains possible solutions to the conflict.

To solve conflicts, POST the "schedule" and "solutions" objects to /api/v1/shows/{show_pk}/schedules/ or PUT to /api/v1/shows/{show_pk}/schedules/{schedule_pk}/ with "solutions" containing values of solutionChoices. Any other value will produce an error.

As long as there's an error, the whole data structure is returned and no database changes will occur. If resolution was successful, database changes take effect and the schedule is returned.

A Schedule is only created/updated if at least one timeslot was created during the resolution process.

Maximum possible output:

```

"solutions": [
  "theirs",
  "ours",
  "theirs-start",
  "ours-start",
  "theirs-end",
  "ours-end",
  "theirs-both",
  "ours-both"
]

```

"theirs" (always possible)

- Discard projected timeslot
- Keep existing timeslot(s)

"ours" (always possible)

- Create projected timeslot
- Delete existing timeslot(s)

"theirs-start"

- Keep existing timeslot
- Create projected timeslot with start time of existing end

"ours-start"

- Create projected timeslot
- Change end of existing timeslot to projected start time

"theirs-end"

- Keep existing timeslot
- Create projected timeslot with end of existing start time

"ours-end"

- Create projected timeslot
- Change start of existing timeslot to projected end time

"theirs-both"

- Keep existing timeslot
- Create two projected timeslots with end of existing start and start of existing end

"ours-both"

- Create projected timeslot
- Split existing into two:
 - Set existing end time to projected start
 - Create another timeslot with start = projected end and end = existing end

Multiple collisions

If there's more than one collision for a projected timeslot, only "theirs" and "ours" are currently supported as solutions.

Errors

Possible error messages are:

Fatal errors that require the schedule's data to be corrected and the resolution to restart

- "Until date mustn't be before start": Set correct start and until dates.
- "Start and until dates mustn't be the same" Set correct start and until dates. (Exception: Single timeslots with recurrence rule 'once' may have the same dates)
- "Numbers of conflicts and solutions don't match": There probably was a change in the schedule by another person in the meantime.
- "This change on the timeslot is not allowed." When adding: There was a change in the schedule's data during conflict resolution. When updating: Fields start, end, byWeekday or rrule have changed, which is not allowed.

Conflict-related errors which can be resolved for each conflict:

- "No solution given": The solutions value was empty or does not exist. Provide a value of solutionChoices
 - "Given solution is not accepted for this conflict.": The solution has a value which is not part of solutionChoices. Provide a value of solutionChoices (at least "ours" or "theirs")
-

3.4.4 Default hosts and ports

Here you find the default hosts and ports for all AURA applications.

Development environment

Component	Host:Port	Description
steering	localhost:8080	Django Development Server
dashboard	localhost:8000	VueJS Development Server
dashboard-clock	localhost:5000	Svelte Development Server
tank	localhost:8040	Go Development Server
engine-api	localhost:8008	Werkzeug Development Server
engine-core	localhost:1234	Liquidsoap Telnet Server
play	localhost:5000	Svelte Development Server

RELEASE NOTES

Here you find the release notes including all changes.

4.1 [1.0.0-alpha4] - 2024-04-18

The release of *Alpha 4 — Raving Raccoon* features:

- **Permission Management:** Easily manage permissions for roles such as Programme Managers and Hosts, as well as individual users.
- **Radio Station Settings:** Administrators now have the ability to configure global station settings conveniently within the new admin area.
- **Playout Refactoring:** The *Engine* component has undergone significant improvements. It now caches data efficiently without the need for a database server. Programme data is cached transparently using JSON files. Additionally, the scheduling and programme polling logic has been refactored and validated using OpenAPI interfaces.
- **Improved Test Coverage:** We have substantially increased test coverage in both *Steering* and *Engine* components. Most components now provide badges displaying the overall test coverage.

Breaking Changes: All nested API endpoints in *Steering* and *Tank* that were deprecated in the previous release have now been removed.

Explore further details on all additional changes below.

4.1.1 Added

- aura: New fixture `radiosettings.json` for defining radio station settings.
- aura: User Guide: Documentation on radio station settings.
- aura: Admin Guide: Info on running Docker Engine as a daemon in production.
- aura: User Guide: Documentation on data visibility, permission and roles in radio administration.
- aura: New Gitlab issue templates for *Bug Report* and *Epic*.
- aura: New “at a glance” overview for release workflow.
- steering: Playlist model as placeholder for custom permissions.
- steering: The route for `schedules` can now handle PATCH requests that don’t involve conflict resolution (steering#218)
- steering: `RadioSettings` model, admin, serializer, viewset and route to expose them (steering#173)

- steering: Management command and Makefile target to delete OIDC client (steering#219)
- dashboard: Implemented basic permissions so that unprivileged users either can't see or can't edit data they are not allowed to see or edit.
- dashboard: Episodes that the dashboard detects as invalid (e.g. those with a negative duration) are now highlighted with a yellow background and red stripes and should be reported.
- dashboard: The menu now has a link to the steering administration page.
- engine: Lots of test cases.

4.1.2 Changed

- aura: Admin Guide: More details on configuring reverse proxies.
- aura: User Guide: Manage groups improved.
- aura: User Guide: Extended "User and host profile" documentation.
- steering: The image URLs are now exposed as `url` and include the protocol to avoid mixed-media warnings (steering#194)
- steering: The `title` of a `Note` is nullable (steering#203)
- steering: A `Note` is attached to a `Timeslot` upon creation (steering#214)
- tank: The job-timeout for the importer is now thirty minutes for Docker.
- dashboard: Calendar event color scheme has been re-worked to be more readable and distinguishes between owned and other shows.
- dashboard: Reworked calendar dialogs and grouped related information and actions.
- dashboard: Drastically reduced asset sizes and therefore improved app load times.
- dashboard: The *Delete all following episodes*-functionality has been temporarily disabled.
- dashboard: The show quick selector has been removed because we provide similar functionality on the show overview page.
- engine: Engine now requires at least Python 3.11.
- engine: Major refactoring of the scheduling mechanics. As a consequence the database dependency for caching is removed. Instead caching is performed in form of `json` files located in the configured `cache_dir` folder.

4.1.3 Removed

- aura: All database related configuration got removed from Engine. Compare and update your `.env` file.
- aura: User Guide: Safari support is removed once again.
- steering: Nested routes for shows, show/timeslots, show/schedules, show/schedule/timeslot (aura#126)
- tank: Nested routes for show/files, show/files and show/playlists (aura#126)
- engine: All database related settings and dependencies. When updating from an existing deployment, remember to compare the new config files and remove relevant entries.

4.1.4 Fixed

- aura: AURA Web health-checks updated to work with Docker Engine 26+.
- dashboard: Fixed some visual spacing issues with dialogs.
- dashboard: The title column in the episode list will now always display a title or placeholder.
- dashboard: Users can now always abort the conflict resolution mode in the calendar.
- dashboard: Fixed some rendering issues for certain conflict resolutions in the calendar.
- dashboard: Fixed some missing, misleading and invalid translations in the calendar.
- engine: Issue where the configuration values for `retry_delay` and `max_retries` are mixed up.

4.1.5 Security

- engine: The codebase is now audited using the security tool [Bandit](#).

4.2 [1.0.0-alpha3] - 2024-02-29

The release of **Alpha 3** — *Playful Platypus* marks a significant milestone with a comprehensive overhaul of the user interface and the play-out stack, introducing many eagerly anticipated features.

The dashboard has undergone a complete redesign in appearance, navigation and functionality. Notable additions in the show area include dialogs for managing show settings, episodes, image uploads, extended metadata assignments, assignment of media sources, and more.

The play-out engine now leverages the state-of-the-art audio server *PipeWire*, providing users with low-latency audio routing options for a smooth playback experience. *PipeWire* is seamlessly integrated into various recent Linux distributions such as *Debian 12* and *Ubuntu 22.04*.

To facilitate understanding of the new features, the [User Guide](#) has been enriched with additional information. Simultaneously, the [Administration Guide](#) has undergone enhancements to expedite the *AURA* setup process.

Additional features include data backup and restoration capabilities, improved logging, and simplified importing of default datasets.

Breaking Changes: Many API implementations are improved to follow best practices. This includes the naming of parameters and properties. The approach how uploaded audio files are stored in the database and on the filesystem got revised. Lastly, most installation steps and their documentation got reworked. Due to the amount of breaking changes we recommend re-installing *AURA* from scratch.

Deprecated: We want to remind you, that nested API endpoints in *Steering* and *Tank* are deprecated. They will be removed with the release of Alpha 4. Try to migrate to the non-nested version of an endpoint and report back if you are facing issues. For more details, check out [api.aura.radio](#) and the detailed changelog below.

Security: Upgrade to [Docker Engine 25.0.2](#) or later due to a [security vulnerability in runc](#).

4.2.1 Added

- aura: CLI command `make version` to display the current version.
- aura: Documentation: Configure playout using PipeWire.
- aura: Documentation: Many chapters in the user documentation on the new features in the show and episode user interfaces.
- aura: New make commands to create sample fixtures and import fixtures (`aura-web.fixtures-create`, `aura-web.fixtures-import`).
- aura: New make commands to backup and restore Docker Volumes (`make aura-volume.backup`, `make aura-volume.restore`).
- aura: Documentation: Chapter to learn how to [backup config and data](#).
- aura: Documentation: Aura Web now features a Quick Install Guide.
- aura: Ability to [load system settings with sample and custom fixtures](#).
- steering: `owners` optional field in `Host` model as many-to-many reference to `User` (steering#166)
- steering: `language` and `topic` optional fields in `Note` model (steering#169)
- steering: `contributors` field in `Note` model (steering#159)
- steering: `PRIVILEGED_GROUP` and `ENTITLED_GROUPS` settings to implement groups and permissions (steering#177)
- steering: `FILTER_ACTIVE_SHOWS_USING` settings to filter active shows and OIDC scope claims (steering#175)
- steering: `ShowManager` to annotate a show with its max (last) timeslot start (steering#175)
- steering: `is_active` boolean field in `LinkType` model (steering#187)
- steering: `addtimeslots` management command to renew schedules for one year (steering#108)
- tank: Un-nested routes for playlists, files, imports, etc. (tank#65)
- dashboard: Added license selection in image picker.
- dashboard: Added show overview page with grid and table view including pagination, search and extensive ordering capabilities.
- dashboard: The state of save operations is now communicated to users.
- dashboard: The media page has been removed in favor of inline editors in the episode and show management pages.
- dashboard: Hierarchical pages now display a breadcrumb navigation.
- dashboard: Show pages now display update/create metadata.
- dashboard: All pages and subviews now have unique and shareable URLs.
- dashboard: A new login page has been added.
- dashboard: Shows can now be deleted.
- dashboard: The show slug can now be changed.
- dashboard: Languages and topics can now be set for individual episodes/notes.
- dashboard: Show on-air and next-up info for episodes.
- dashboard: Added support for viewing past episodes.
- dashboard: Added support for viewing inactive schedules.

- dashboard: Platform and social media links can now be added to shows and episodes.
- dashboard: Content fields now have a WYSIWYG-HTML editor.
- dashboard-clock: Sample API data for testing provided via `/test/api.json`
- engine: Additional logging while loading playlist entries (#136)
- engine-api: make api: generate, merge and clean files according to api specs in `schemas/openapi-engine.yaml` (#37)
- engine-core: PipeWire as the media server inside the docker container
- engine-core: Docker compose support for easier building of local containers
- engine-core: WirePlumber script to automatically connect ports
- engine-core: WirePlumber script to list device ports
- engine-recorder: Added a option to configure the sub directories where the audio files are saved (engine-recorder#36)
- engine-recorder: Added local docker build (engine-recorder#31)
- engine-recorder: Added docker compose

4.2.2 Changed

- aura: Documentation: Add details on how to release a service in the developer docs.
- aura: **BREAKING:** Update naming and meaning of several make commands and according documentation. Re-read the docs carefully when installing.
- aura: Docker images are now tagged and published with `main` instead of `unstable`.
- aura: The aura user is now created as a system user.
- aura: Documentation: Multiple content and structure improvements in the Administration Guide.
- aura: Documentation: CertBot in Aura Web installation is now enabled by default. Disable, when not needed.
- aura: Sample configuration override for `engine-api` is now in `YAML` format, instead of `INI`.
- aura: Update `engine-recorder` sample configuration for override.
- aura: Nginx is running rootless now. This improves security and fixes some other permission bugs.
- aura: Replace `AURA_ENGINE_RECORDER_AUDIO_DEVICE` with `AURA_ENGINE_RECORDER_AUDIO_SOURCE` and `AUDIO_DEVICE` with `AUDIO_SOURCE`, (engine-recorder#32)
- steering: **BREAKING:** The IDs of the “owned shows” and the “public shows” are now listed as `ownedShowIds` and `publicShowIds` in the `OICD` scope claims.
- steering: `note_id` and `show_id` are read-only fields of the `TimeslotSerializer` (steering#171)
- steering: `Note.tags` are exposed as list of strings (steering#158)
- steering: Django’s model permissions and additional custom permissions are now used (steering#177)
- steering: `type` in `Link` model is now foreign key reference to the `LinkType` model. The serializers expect a `typeId` (steering#187)
- tank: **BREAKING:** The database and directory structure has changed. Since there is no way to migrate the data for this update, you need to re-create the database and delete the existing files and folders from the audio store.
- tank: **BREAKING:** The endpoints to create files and playlists now require a `showId` in the `JSON` payload.

- tank: **BREAKING:** API endpoints now require `showId` (int) instead of the `showName` (string).
- tank: The duration is now in seconds and is stored as a float.
- tank: The job-timeout for the importer is now three minutes for Docker.
- tank: Show uploads are stored in folders named after the ids instead of the slugs. (tank#37)
- tank: error and info log files now have timestamps (tank#68)
- tank: The endpoint to upload a file with Flow JS no longer requires a `showId`.
- tank: The endpoint to list the playlists now lists all, the ones by a show or the ones that include a file.
- dashboard: The global navigation has been moved to a sidebar.
- dashboard: The global navigation now includes contextualized submenus.
- dashboard: The show admin selector now uses an inline selection mechanism instead of a dialog.
- dashboard: The help link in the footer now directly links to the AURA documentation.
- dashboard: The page-size selection for the episode table has been moved to the table footer.
- dashboard: The range-end selection in the episode table has been removed.
- dashboard: The range-start selection in the episode table is now positioned right next to the table header.
- dashboard: The episode/note editor is now a separate page instead of a modal.
- dashboard: The show deactivation button has been moved to the *Danger zone* and now requires confirmation.
- dashboard: The show page has been split into two separate pages for episodes/schedules and basic settings.
- dashboard: Related individual fields in the basic show settings have been restructured into structured blocks.
- dashboard: Updated/harmonized terms used in show pages.
- dashboard: The unimplemented settings page has been removed.
- engine: Configuration using yaml instead of ini files (#120)
- engine: Change default database password to string to avoid parsing errors (#120)
- engine: Use datatype float instead of int for `trackDuration` (#132)
- engine-api: Configuration using yaml instead of ini files (#34)
- engine-api: Use datatype float instead of int for `trackDuration` (#51)
- engine-api: Change default database password to string to avoid parsing errors (#54)
- engine-core: Default sound system is now JACK, not ALSA
- engine-core: Docs: Update README with new PipeWire based installation
- engine-recorder: Renamed the `input.file` field to `input.source` (engine-recorder#32)
- engine-recorder: Renamed the `input.file_format` field to `input.source_format` (engine-recorder#32)
- engine-recorder: Replaced `AURA_ENGINE_RECORDER_AUDIO_DEVICE` with `AURA_ENGINE_RECORDER_AUDIO_SOURCE` and `AUDIO_DEVICE` with `AUDIO_SOURCE`, (engine-recorder#32)

4.2.3 Deprecated

- aura: Nested API endpoints in Steering and Tank are deprecated and will be removed in the next release. Compare the specification at api.aura.radio and update your API client as soon as possible. Please report if you are facing issues.
- steering: Nested routes for shows, show/timeslots, show/schedules, show/schedule/timeslot will be removed with alpha-4.
- tank: Nested routes for show/files and show/playlists will be removed with alpha-4.

4.2.4 Removed

- steering: `note_id` and `show` fields from the Timeslot model (steering#171)
- steering: `type` field from LinkType model (steering#187)
- steering: `owner` and `slug` fields from Note model.
- engine-api: Remove endpoint `/playlog/report/{year_month}` (#48)
- engine-core: Dropped ALSA in Docker support in favour of PipeWire + JACK

4.2.5 Fixed

- aura: To avoid permission problems re-run the commands `make aura-web.init` or `make aura-playout.init` respectively.
- aura: Warning when executing the make target for adding the aura user.
- aura: Consolidated the structure for custom configuration and fixtures in AURA Web.
- aura: Due to certificate retrieval issues, certbot is now running on port 80 (#276).
- aura: Missing image in the documentation under a [administration / update and maintain](#).
- aura: Allow optional trailing slash for `/trackservice` endpoints and don't redirect (engine-api#25).
- aura: Documentation: Replace `AURA_RECORDER_AUDIO_STORE_HOST` with `AURA_AUDIO_STORE`, (engine-recorder#39)
- dashboard: Fixed overflow issues for in image picker.
- dashboard: Ensure save errors due to connection issues are properly communicated to the user.
- dashboard: Fixed keyboard selection support in image picker.
- dashboard: Fixed overflow issues in calendar.
- dashboard: Fixed overflow issues in show quick selector for long show titles.
- engine: Fix redundant logging while loading playlist entry (#136)
- engine: Fix line-in source selection (#141)
- engine-core: Smooth play-out latency and buffering for analog audio input (engine-core#50)
- engine-recorder: Fixed a bug where the audio files were always saved with the `_%s` suffix (engine-recorder#35)

4.2.6 Security

- aura: Upgrade to [Docker Engine 25.0.2](#) or later due to security vulnerability in `runc`.

4.3 [1.0.0-alpha2] - 2023-06-29

The release of **Alpha 2 — *Precise Pheasant*** ships with plenty of essential improvements, both under the hood and on the user interface level.

The *Data Model* has been extended and refined to provide a foundation for the requirements of participating radios. Additional *recurrence rules* have been added as part of the default configuration.

The *User Interface* has undergone a facelift and has been updated with features and enhancements, especially in the calendar and show view.

Administrators are now served with log files for all services. In order to avoid permission errors, also re-run the `make aura-user.add` command. This now creates directories for `logs` and `audio` with correct permissions for the `aura` user.

Breaking Changes:

- Some configuration settings for the *Docker Compose* bundles have been changed. Either adapt your current `.env` file accordingly or start with a fresh one.
- The API has been improved to follow best practices. This includes the naming of parameters and properties.

Deprecated:

- Nested API endpoints in *Steering* and *Tank* are now deprecated. Try to migrate to the non-nested version of an endpoint and report back if you are facing issues.

For more details, check out [api.aura.radio](#) and the detailed changelog below.

4.3.1 Added

- aura: Add development target `make dev.merge-changelog` to create a merged changelog, helping to prepare release notes.
- aura: Add release notes section at [docs.aura.radio](#)
- aura: Add Tank Cut & Glue API to [api.aura.radio](#).
- aura: Add default [Audio Store](#) directory `/var/audio/import` used for audio file imports via filesystem (aura#172).
- aura: Add default setting for `AURA_AUDIO_STORE_RECORDINGS` to the `sample.env` file.
- aura: Add audio targets for the Makefile to start and disable PulseAudio.
- aura: Add optional engine-recorder profile to `aura-playout`.
- aura: Add an log file for NGINX, `aura-web` (#147).
- aura: Documentation: Add more details to the [Radio Station Administration](#) chapter.
- aura: Documentation: Add chapter on deleting and deactivating users in the User Guide.
- aura: Documentation: Add more info on switching between releases to the Administration Guide.
- aura: Add directories to `logs` and `audio` to avoid permission errors (#209)
- steering: Image concrete model to handle all instances.

- steering: API endpoint `/api/v1/images/` to add, update and delete images.
- steering: Add Host admin
- dashboard: a new show selector to the navbar (#122, f9762d85).
- dashboard: a new image picker that can be used to upload new and select existing images (#89, #139, e0fd0a76, b335d801, 57e1e10f).
- dashboard: ability to add internal notes to a show (#146, 319cf540).
- dashboard: information on empty slots between shows and a progress indicator for the currently playing show
- dashboard: to the calendar day-view (#155, 9ac48a55, 2a68a7ac).
- dashboard: a progress indicator for the currently playing show to the calendar week-view (#156, b52672dd).
- dashboard: ability to add and remove individual contributors to the show notes based on the show owners (#170, #146, aa707763).
- dashboard: ability to add tags to show notes (#146, 560d8dda).
- engine: API responses from Steering and Tank are now cached in `cache_dir`
- engine-api: Test coverage (`make coverage` #40)
- engine-api: Badge displaying coverage result (#40)

4.3.2 Changed

- aura: Update sample configuration files for services in `config/service/sample-config`.
- aura: Rename environment variable `AURA_ENGINE_SERVER_TIMEOUT` to `AURA_ENGINE_SERVER_TIMEOUT` in `sample.env`.
- aura: Increase default value for `AURA_ENGINE_SERVER_TIMEOUT` in `sample.env`, to avoid malfunctions when idle (aura#165).
- aura: Documentation: Update Contribution Guidelines and Coding Conventions in the Developer Guide.
- aura: Documentation: Extend section for developers on performing releases.
- aura: Change `/trackservice` endpoints to `/engine/api/v1/trackservice` and `/engine/api/v1/trackservice/current` according to API namespace conventions (aura#190).
- steering: The `Host`, `Note`, `Show` models & serializers reference the new `Image`.
- steering: The `logo` field in the `Show` model is a reference to `Image`.
- steering: The “conflict resolution” methods of `Schedule` are now service functions.
- steering: Update all APIs to return attributes / properties in camelCase notation (aura#141)
- steering: Use `_id` suffix for all object reference in REST APIs (aura#166)
- steering: Use blank strings instead of nullable strings in REST APIs (aura#167)
- steering: Upgrade Poetry dependencies and Django to the next LTS version (steering#137)
- dashboard: The timeslot overview has been fully revised (4ef88279).
- dashboard: The timeslot show notes editor has been fully revised (#141, 57e1e10f).
- dashboard: Past schedules are now hidden in the show manager (#120, 8cd743de).
- dashboard: Schedules are now shown with their respective start and end date (#121, f9dc6fef).
- dashboard: The calendar day-view has been fully revised (#155, 9ac48a55, 2a68a7ac).

- dashboard: The slots in the calendar week-view have been from 30 to 15 minutes to make it easier to
- dashboard: create shorter slots (#151, 3fc47383).
- dashboard: Very short timeslots in the calendar week-view now take up less space (#151, 3fc47383, bfe7f813).
- dashboard: Show start and end time in calendar tooltip if the event is too short to render it as content (#173, cf5ac12e).
- dashboard: The show manager interface for editing the show details has been overhauled, changing the overall appearance
- dashboard: and the way some of the fields are edited (db7dc49d).
- dashboard-clock: Provide properties in API schemas in CamelCase notation (aura#141)
- engine: Provide properties in API schemas in CamelCase notation (aura#141)
- engine-api: Make properties in API schemas in CamelCase notation (aura#141)
- engine-api: Avoid deprecation warning by replacing JSON encoder (#35)
- engine-api: Change HTTP status codes (204 No Content) for successful but empty POST/PUT responses, adapt tests (#5)
- engine-core: Make properties in API schemas in CamelCase notation (aura#141)
- engine-core: Configuration: Renamed environment variable `AURA_ENGINE_SERVER_TIMEOUT` to `AURA_ENGINE_SERVER_TIMEOUT`
- engine-core: and configuration setting `telnet_server_timeout` to `server_timeout`.
- engine-core: Configuration: Increase default value for `server_timeout`, to avoid malfunctions when idle (aura#165)

4.3.3 Deprecated

- aura: Nested API endpoints in Steering and Tank are now deprecated. Compare the specification at api.aura.radio

4.3.4 Removed

- steering: The abstract `ModelWithImageFields` from the program models.
- steering: The `ThumbnailsMixin` from the program serializers.
- steering: The abstract `ModelWithCreatedUpdatedFields` from the program models.
- dashboard: Non-functional nav button to broadcast management on show detail page (#133, e4083e66).
- engine: Remove mail service as it will be replaced by Prometheus monitoring (engine#109)

4.3.5 Fixed

- aura: Fix an issue where `make aura-user .add` causes an unrelated Python error, raised from another command (#183).
- aura: Fix the location of the Docker Compose bundles in the README.
- aura: Fix an issue where `/trackservice` was not reachable (#159).
- aura: Fix an issue where some settings in the `.env` and `.yml` were not applied correctly (#160).
- aura: Fix an issue with the `aura-playout` targets in the Makefile (#168).
- aura: Documentation: Replace old “meta” repository references with “aura”.
- steering: use kebab-case in URLs
- steering: don’t output invalid PPOI format
- steering: note image should allow null values
- steering: don’t force REST API consumers to set `repetition_of` for timeslots
- steering: The timeslot generation leaves the `last_date` unset if it is null.
- steering: Note.tags attribute not updated via API (steering#157)
- steering: make subtitle field in Category a CharField
- dashboard: Navigation would sometimes freeze, after an application error in the playlists view (#132, e4083e66).
- dashboard: Calendar would sometimes crash if playlist data is missing (#132, a3e8e3b0).
- dashboard: Switching between the week and day-view in the calendar no longer squishes the week-view (#153, e28eb4f7).
- dashboard: The calendar uses entire viewport height now (#154, 4c90f277).
- dashboard: Users are now able to create a timeslot in a calendar week slot that has already started (#167, 3d4e9b28).
- dashboard: HTML in the show title, short description and description is now correctly rendered (#129, 649b4c0b).
- engine: Fix an issue where the Engine version is displayed in the logs wrongly
- engine-api: Fix an issue where the configuration file is overwritten in the `make init.dev` target
- engine-api: Fix an issue where the Docker Compose healthcheck failed (#39)
- engine-api: Fix broken endpoint `/trackservice` (aura#185)
- engine-core: Telnet server sample config not working (engine-core#45).
- engine-core: Extend and improve output for `make help`.
- engine-recorder: Fixed a bug where the recorder would not start when trying to record a web stream (aura#182)

4.4 [1.0.0-alpha1] - 2023-03-02

Initial release.

BUG REPORTS

Tell us about your problems!

But please use the **defect template** as described below to submit your bug reports. Or get in touch directly.

5.1 Contact us via Matrix

You can find us on [Matrix](#) which is also our primary channel for communication.

5.2 Create a ticket on GitLab

If you don't have a GitLab account, you can [sign up here](#).

1. Head over to our GitLab instance at [code.aura.radio](#).
2. Search across all projects for existing tickets relevant to your issue.
3. If something is available already, vote for it and place your comment.
4. If nothing is available, first choose the relevant project. If you are unsure about the project, use the [aura repository](#). Then file a new ticket by clicking the button *New issue*.
5. Now click the dropdown *Choose a template* and select the *Bug Report* template.
6. Fill out all the relevant sections and submit the defect ticket.

CONTRIBUTING TO AURA

6.1 Code of Conduct

We inherit the *Contributor Covenant*.

6.2 How can I contribute?

You don't need to be a developer to contribute to the project.

- Join the [AURA Matrix Space](#).
- Check out the source code and try AURA for yourself.
- *Create bug reports, feature requests and provide thoughts in GitLab.*
- Become an active developer or maintainer. To do so, check out the *Developer Guide*, especially the *Coding Conventions*.
- Provide sponsorship. We are happy to list you on the front page.

6.3 Contribution Guidelines

- Read up on our *Developer Guide > Coding Conventions*.
- When contributing code, create a *Merge Request* and assign the repository maintainer for review.
- We write changelogs in the [keep a changelog](#) format. Update the `CHANGELOG.md` before committing code. Remember that changelogs are meant for humans and should reflect the end-user value.
- In case of big changes affecting several services and APIs it might be good to discuss that with the team beforehand. Think about doing a test deployment with all required services.

6.4 Contributors

- Code contributors can be found in the `git` logs.
- Martin Lasinger from Freies Radio Freistadt designed the AURA logos and icons.
- The foundation of Steering is based on [Radio Helsinki's PV Module](#) by Ernesto Rico Schmidt.
- The foundation of Engine is based on [Comba](#), by Michael Liebler and Steffen Müller.

6.5 Partners and Sponsorship

Current partners, sponsors and supporters

- [Radio Orange 94.0 - Verein Freies Radio Wien](#)
- [Radio Helsinki - Verein freies Radio Steiermark](#)
- [Radio FRO - Freier Rundfunk Oberösterreich](#)
- [Freies Radio Wüste Welle](#)
- [Freies Radio Freistadt](#)
- [Radio free FM](#)
- [FREIRAD Freies Radio Innsbruck](#)
- [Servus.at - Kunst & Kultur im Netz](#)

Previous partners and sponsors

- [Radiofabrik - Verein Freier Rundfunk Salzburg](#)

6.6 Licensing

By contributing your code you agree to these licenses and confirm that you are the copyright owner of the supplied contribution.

6.7 License

- Logos and trademarks of sponsors and supporters are copyrighted by the respective owners. They should be removed if you fork this repository.
- All source code is licensed under [GNU Affero General Public License \(AGPL\) v3.0](#).
- All other assets and text are licensed under [Creative Commons BY-NC-SA v3.0](#).

These licenses apply unless stated differently.

CONTRIBUTOR COVENANT CODE OF CONDUCT

7.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, caste, color, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

7.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

7.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

7.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

7.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at [aura-dev \(at\) subsquare.at](mailto:aura-dev@subsquare.at). All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

7.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

7.6.1 1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

7.6.2 2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

7.6.3 3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

7.6.4 4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

7.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.1, available at https://www.contributor-covenant.org/version/2/1/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at <https://www.contributor-covenant.org/faq>. Translations are available at <https://www.contributor-covenant.org/translations>.

7.8 License

Contributor Covenant is released under the [Creative Commons Attribution 4.0 International Public License](https://creativecommons.org/licenses/by/4.0/).

```
[meta]: Hello, welcome, this is meta!
[you]: Oh, hi meta, what are you meta about?
[meta]: I am the place where all the work behind the 'real' work is happening.
[meta]: I collect stuff about the AuRa project and greet new visitors.
[you]: Ah, kool. Well... I am a new visitor, what can you tell me about this?
[meta]: **loading project overview info**
[meta]: Hello new visitor, here you go:
```


ABOUT AURA

What is this about? You probably already know. If not, here are some links that explain in more detail what the community radio stations in .AT land and .DE land are which call themselves *Freie Radios* (literally it would be translated to *free radios*, but as the thing with *free* here is the same as with *free software* in general).

Unfortunately most of those links are in German language as our constituency is located primarily in Austria and German. We will provide a short intro in English language as soon as we find some time.

About *Freie Radios*:

- <http://freie-radios.at> - Austrian association of community radio stations
- <http://freie-radios.de> - German association of community radio stations
- <https://cba.fro.at> - Podcast platform of Austrian community radio stations
- <https://freie-radios.net> - Audio portal of German community radio stations
- <https://amarceurope.eu> - European association of community radio stations

And what is this here now?

AuRa is a suite of radio management, programme scheduling and play-out automation software that fits the needs of free radio stations. The initiative took off in Austria, where several stations are still using and depending on *Y.A.R.M.* (which is just yet another radio manager). *Y.A.R.M.* was an awesome project that provided a software which was tailored to how community radio stations create their diverse programmes. Unfortunately it is also a piece of monolithic Java code that has come into the years. Also it never really took off as a free software project and was depending on a single developer. Today nobody really wants to touch its code anymore.

Now we urgently need something new, and all those other solutions out there (FLOSS as well as commercial) do not really fulfill all our requirements. Therefore we decided to pool our resources and develop something new, while reusing a lot of work that has already been done at one or another station.

MATRIX

You can get in touch and contribute via [Matrix](#) and [Element](#). This is our main communication channel.

Join the discussion at our Matrix Space and its main channels:

- [AURA Matrix Space](#)
 - [AURA-general](#)
 - [AURA-dev](#)
 - [AURA-support](#)
 - [AURA-notifications](#)
 - [AURA-offtopic](#)

MAILINGLIST

We sporadically share newsletters at the `users (at) aura.radio` Mailinglist.

10.1 Subscribing to the list

In order to subscribe send a mail with following content to `majordomo (at) aura.radio`:

```
subscribe users-aura-radio
```

Then follow the instructions for confirmation in the reply mail. Alternatively you can message the development team directly via `dev (at) aura.radio`, and we will add you to the list.

10.2 Unsubscribing from the list

In order to unsubscribe send this to `majordomo (at) aura.radio`:

```
unsubscribe users-aura-radio
```

CHAPTER
ELEVEN

PARTNERS