

---

# Alliance Auth Documentation

*Release*

**R4stl1n**

**Apr 03, 2020**



<b>1</b>	<b>Installing</b>	<b>3</b>
<b>2</b>	<b>Using</b>	<b>5</b>
<b>3</b>	<b>Troubleshooting</b>	<b>7</b>
3.1	Features . . . . .	7
3.1.1	The State System . . . . .	7
3.1.2	Groups . . . . .	8
3.1.3	Group Management . . . . .	10
3.1.4	Auto Groups . . . . .	12
3.1.5	HR Applications . . . . .	14
3.1.6	Corp Stats . . . . .	16
3.1.7	Permissions Auditing . . . . .	22
3.1.8	Services Name Formats . . . . .	24
3.1.9	Fleetup . . . . .	25
3.1.10	Fleet Activity Tracking . . . . .	26
3.1.11	Optimizer . . . . .	26
3.1.12	SRP . . . . .	26
3.1.13	Timerboard . . . . .	26
3.2	Installation . . . . .	26
3.2.1	Auth . . . . .	26
3.2.2	Services . . . . .	39
3.3	Maintenance . . . . .	61
3.3.1	Your Auth Project . . . . .	61
3.3.2	Troubleshooting . . . . .	64
3.4	Development . . . . .	65
3.4.1	Documentation . . . . .	65
3.4.2	Integrating Services . . . . .	66
3.4.3	Menu Hooks . . . . .	72
3.4.4	URL Hooks . . . . .	73



An auth system for EVE Online to help in-game organizations manage online service access.



---

## Installing

---

*Setup Guide*





---

## Using

---

Learn about individual *features*.



---

## Troubleshooting

---

Read the *list of common problems*.

### 3.1 Features

#### 3.1.1 The State System

##### Overview

In Alliance Auth v1 admins were able to define which Corporations and Alliances were to be considered “members” with full permissions and “blues” with restricted permissions. The state system is the replacement for these static definitions: admins can now create as many states as desired, as well as extend membership to specific characters.

##### Creating a State

States are created through your installation’s admin site. Upon install three states are created for you: Member, Blue, and Guest. New ones can be created like any other Django model by users with the appropriate permission (`authentication | state | Can add state`) or superusers.

A number of fields are available and are described below.

##### Name

This is the displayed name of a state. Should be self-explanatory.

##### Permissions

This lets you select permissions to grant to the entire state, much like a group. Any user with this state will be granted these permissions.

A common use case would be granting service access to a state.

### Priority

This value determines the order in which states are applied to users. Higher numbers come first. So if a random user Bob could member of both the `Member` and `Blue` states, because `Member` has a higher priority Bob will be assigned to it.

### Public

Checking this box means this state is available to all users. There isn't much use for this outside the `Guest` state.

### Member Characters

This lets you select which characters the state is available to. Characters can be added by selecting the green plus icon.

### Member Corporations

This lets you select which Corporations the state is available to. Corporations can be added by selecting the green plus icon.

### Member Alliances

This lets you select which Alliances the state is available to. Alliances can be added by selecting the green plus icon.

### Determining a User's State

States are mutually exclusive, meaning a user can only be in one at a time.

Membership is determined based on a user's main character. States are tested in order of descending priority - the first one which allows membership to the main character is assigned to the user.

States are automatically assigned when a user registers to the site, their main character changes, they are activated or deactivated, or states are edited. Note that editing states triggers lots of state checks so it can be a very slow process.

Assigned states are visible in the `Users` section of the `Authentication` admin site.

### The Guest State

If no states are available to a user's main character, or their account has been deactivated, they are assigned to a catch-all `Guest` state. This state cannot be deleted nor can its name be changed.

The `Guest` state allows permissions to be granted to users who would otherwise not get any. For example access to public services can be granted by giving the `Guest` state a service access permission.

## 3.1.2 Groups

Group Management is one of the core tasks of Alliance Auth. Many of Alliance Auth's services allow for synchronising of group membership, allowing you to grant permissions or roles in services to access certain aspects of them.

## User Organised Groups

Administrators can create custom groups for users to join. Examples might be groups like Leadership, CEO or Scouts.

When you create a Group additional settings are available beyond the normal Django group model. The admin page looks like this:

Home · Group Management · Groups · Recruiters

Change group
HISTORY

Name: Recruiters

Permissions:

Available permissions ⓘ

Q
Filter

admin | log entry | Can add log entry  
admin | log entry | Can change log entry  
admin | log entry | Can delete log entry  
auth | group | Can add group  
auth | group | Can change group  
auth | group | Can delete group  
auth | permission | Can add permission  
auth | permission | Can change permission  
auth | permission | Can delete permission  
auth | user | Can add user  
auth | user | Can change user

Choose all ⓘ

Chosen permissions ⓘ

auth | user | human\_resources

Remove all

Hold down "Control", or "Command" on a Mac, to select more than one.

AUTH SETTINGS

: Recruiters

Description: People who screen applications.

Description of the group shown to users.

Group leaders:

AVAILABLE GROUP LEADERS ⓘ

Q
Filter

Choose all ⓘ

CHOSEN GROUP LEADERS ⓘ

edemof

Remove all

Group leaders can process group requests for this group specifically. Use the auth.group\_management permission to allow a user to manage all groups. Hold down "Control", or "Command" on a Mac, to select more than one.

☐ Internal

Internal group, users cannot see, join or request to join this group.  
Used for groups such as Members, Corp\_\*, Alliance\_\* etc.  
Overrides Hidden and Open options when selected.

☒ Hidden

Group is hidden from users but can still join with the correct link.

☐ Open

Group is open and users will be automatically added upon request.  
If the group is not open users will need their request manually approved.

☐ Public

Group is public. Any registered user is able to join this group, with visibility based on the other options set for this group.  
Auth will not remove users from this group automatically when they are no longer authenticated.

Delete

Save and add another

Save and continue editing

SAVE

Here you have several options:

3.1. Features

9

### Internal

Users cannot see, join or request to join this group. This is primarily used for Auth's internally managed groups, though can be useful if you want to prevent users from managing their membership of this group themselves. This option will override the Hidden, Open and Public options when enabled.

By default, every new group created will be an internal group.

### Hidden

Group is hidden from the user interface, but users can still join if you give them the appropriate join link. The URL will be along the lines of `https://example.com/en/group/request_add/{group_id}`. You can get the Group ID from the admin page URL.

This option still respects the Open option.

### Open

When a group is toggled open, users who request to join the group will be immediately added to the group.

If the group is not open, their request will have to be approved manually by someone with the group management role, or a group leader of that group.

### Public

Group is accessible to any registered user, even when they do not have permission to join regular groups.

The key difference is that the group is completely unmanaged by Auth. **Once a member joins they will not be removed unless they leave manually, you remove them manually, or their account is deliberately set inactive or deleted.**

Most people won't have a use for public groups, though it can be useful if you wish to allow public access to some services. You can grant service permissions on a public group to allow this behaviour.

### Permission

In order to join a group other than a public group, the permission `groupmanagement.request_groups` (Can request non-public groups in the admin panel) must be active on their account, either via a group or directly applied to their User account.

When a user loses this permission, they will be removed from all groups *except* Public groups.

---

**Note:** By default, the `groupmanagement.request_groups` permission is applied to the Member group. In most instances this, and perhaps adding it to the Blue group, should be all that is ever needed. It is unsupported and NOT advisable to apply this permission to a public group. See #697 for more information.

---

## 3.1.3 Group Management

In order to access group management, users need to be either a superuser, granted the `auth | user | group_management` ( Access to add members to groups within the alliance ) permission or a group leader (discussed later).

## Group Requests

When a user joins or leaves a group which is not marked as “Open”, their group request will have to be approved manually by a user with the `group_management` permission or by a group leader of the group they are requesting.

## Group Membership

The group membership tab gives an overview of all of the non-internal groups.

<div> Group Management Group Requests Group Membership </div>				
Groups				
Name	Description	Status	Member Count	Action
Blackops	Drop it like its hot	Open	2	
Hidden Group		Hidden	0	
Open Group	An Open Group	Open	0	
Requestable Group		Requestable	1	
Test Group 2	It does things	Hidden	0	
Ts3 Test Group		Open	0	

## Group Member Management

Clicking on the blue eye will take you to the group member management screen. Here you can see a list of people who are in the group, and remove members where necessary.

<div> Group Management Group Requests Group Membership </div>				
Blackops Members				
User	Character	Corp	Alliance	Action
basraah	Basraah	Ice Fire Warriors	Escalating Entropy	
basraah3				

## Group Audit Log

Whenever a user Joins, Leaves, or is Removed from a group, this is logged. To find the audit log for a given group, click the light-blue button to the right of the Group Member Management (blue eye) button.

These logs contain the Date and Time the action was taken (in EVE/UTC), the user which submitted the request being acted upon (requestor), the user’s main character, the type of request (join, leave or removed), the action taken (accept, reject or remove), and the user that took the action (actor).

Group ManagementGroup RequestsGroup Membership

Example Group Audit Log

All times displayed are EVE/UTC.

Show 10 entries

Search:

Date/Time	Requestor	Main Character	Group	Type	Action	Actor
Dec. 8, 2018, 11:36 p.m.	root	Col Crunch	Example Group	Removed	Removed	root
Dec. 8, 2018, 11:36 p.m.	root	Col Crunch	Example Group	Leave	Reject	root
Dec. 8, 2018, 11:36 p.m.	root	Col Crunch	Example Group	Join	Accept	root

Showing 1 to 3 of 3 entries

Previous1Next

**Note:** There is no tracking for “Open” groups as members are able to freely join/leave these groups.

## Group Leaders

Group leaders have the same abilities as users with the `group_management` permission, *however*, they will only be able to:

- Approve requests for groups they are a leader of.
- View the Group Membership and Group Members of groups they are leaders of.

This allows you to more finely control who has access to manage which groups. Currently it is not possible to add a Group as group leaders.

### 3.1.4 Auto Groups

**Note:** New in 2.0

Auto groups allows you to automatically place users of certain states into Corp or Alliance based groups. These groups are created when the first user is added to them and removed when the configuration is deleted.

## Installation

Add `'allianceauth.eveonline.autogroups'`, to your `INSTALLED_APPS` list and run migrations. All other settings are controlled via the admin panel under the `Eve_Autogroups` section.

## Configuring a group

When you create an autogroup config you will be given the following options:



## Add autogroups config

**States:**

Member

Blue

Guest

+

Hold down "Control", or "Command" on a Mac, to select more than one.

☐ Corp groups

Setting this to false will delete all the created groups.

Corp group prefix:

**Corp name source:**

Full name

☐ Alliance groups

Setting this to false will delete all the created groups.

Alliance group prefix:

**Alliance name source:**

Full name

☐ Replace spaces

Replace spaces with:

Any spaces in the group name will be replaced with this.

**Warning:** After creating a group you wont be able to change the Corp and Alliance group prefixes, name source and the replace spaces settings. Make sure you configure these the way you want before creating the config. If you need to change these you will have to create a new autogroup config.

- States selects which states will be added to automatic Corp/Alliance groups
- Corp/Alliance groups checkbox toggles Corp/Alliance autogroups on or off for this config.
- Corp/Alliance group prefix sets the prefix for the group name, e.g. if your Corp was called `MyCorp` and your prefix was `Corp`, your autogroup name would be created as `Corp MyCorp`. This field accepts leading/trailing spaces.
- Corp/Alliance name source sets the source of the Corp/Alliance name used in creating the group name. Currently the options are Full name and Ticker.
- Replace spaces allows you to replace spaces in the autogroup name with the value in the Replace spaces with field. This can be blank.

### 3.1.5 HR Applications

#### Installation

Add `'allianceauth.hrapplications'`, to your `INSTALLED_APPS` list in your auth project's settings file. Run migrations to complete installation.

#### Management

##### Creating Forms

The most common task is creating `ApplicationForm` models for corps. Only when such models exist will a Corporation be listed as a choice for applicants. This occurs in the Django admin site, so only staff have access.

The first step is to create questions. This is achieved by creating `ApplicationQuestion` models, one for each question. Titles are not unique.

Next step is to create the actual `ApplicationForm` model. It requires an existing `EveCorporationInfo` model to which it will belong. It also requires the selection of questions. `ApplicationForm` models are unique per Corporation: only one may exist for any given Corporation concurrently.

You can adjust these questions at any time. This is the preferred method of modifying the form: deleting and recreating will cascade the deletion to all received applications from this form which is usually not intended.

Once completed the Corporation will be available to receive applications.

##### Reviewing Applications

Superusers can see all applications, while normal members with the required permission can view only those to their corp.

Selecting an application from the management screen will provide all the answers to the questions in the form at the time the user applied.

When a reviewer assigns themselves an application, they mark it as in progress. This notifies the applicant and permanently attached the reviewer to the application.

Only the assigned reviewer can approve/reject/delete the application if they possess the appropriate permission.

Any reviewer who can see the application can view the applicant's APIs if they possess the appropriate permission.

## Permissions

The following permissions have an effect on the website above and beyond their usual admin site functions.

Permission	Admin Site	Auth Site
auth.human_resources	None	Can view applications and mark in progress
hrapplications.approve_application	None	Can approve applications
hrapplications.delete_application	Can delete model	Can delete applications
hrapplications.reject_applications	None	Can reject applications
hrapplications.view_apis	None	Can view applicant API keys, and audit in Jackknife
hrapplications.add_applicationcomment	Can create model	Can comment on applications

Best practice is to bundle the `auth.human_resources` permission alongside the `hrapplications.approve_application` and `hrapplications.reject_application` permissions, as in isolation these don't make much sense.

## Models

### ApplicationQuestion

This is the model representation of a question. It contains a title, and a field for optional “helper” text. It is referenced by `ApplicationForm` models but acts independently. Modifying the question after it has been created will not void responses, so it's not advisable to edit the title or the answers may not make sense to reviewers.

### ApplicationForm

This is the template for an application. It points at a `Corporation`, with only one form allowed per `Corporation`. It also points at `ApplicationQuestion` models. When a user creates an application, they will be prompted with each question the form includes at the given time. Modifying questions in a form after it has been created will not be reflected in existing applications, so it's perfectly fine to adjust them as you see fit. Changing `Corporations` however is not advisable, as existing applications will point at the wrong `Corporation` after they've been submitted, confusing reviewers.

### Application

This is the model representation of a completed application. It references an `ApplicationForm` from which it was spawned which is where the `Corporation` specificity comes from. It points at a user, contains info regarding its reviewer, and has a status. Shortcut properties also provide the applicant's main character, the applicant's APIs, and a string representation of the reviewer (for cases when the reviewer doesn't have a main character or the model gets deleted).

### ApplicationResponse

This is an answer to a question. It points at the `Application` to which it belongs, to the `ApplicationQuestion` which it is answering, and contains the answer text. Modifying any of these fields is dangerous.

### ApplicationComment

This is a reviewer's comment on an application. Points at the application, points to the user, and contains the comment text. Modifying any of these fields is dangerous.

## Troubleshooting

### No corps accepting applications

Ensure there are ApplicationForm models in the admin site. Ensure the user does not already have an application to these Corporations. If the users wishes to re-apply they must first delete their completed application

### Reviewer unable to complete application

Reviewers require a permission for each of the three possible outcomes of an application, Approve Reject or Delete. Any user with the human resources permission can mark an application as in-progress, but if they lack these permissions then the application will get stuck. Either grant the user the required permissions or change the assigned reviewer in the admin site. Best practice is to bundle the `auth.human_resources` permission alongside the `hrapplications.approve_application` and `hrapplications.reject_application` permissions, as in isolation these don't serve much purpose.

## 3.1.6 Corp Stats

This module is used to check the registration status of Corp members and to determine character relationships, being mains or alts.

### Installation

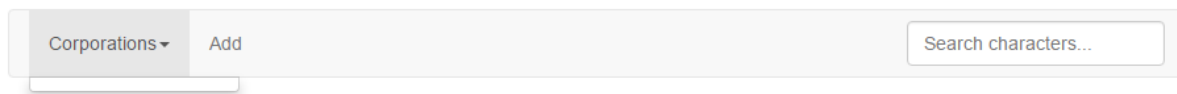
Corp Stats requires access to the `esi-corporations.read_corporation_membership.v1` SSO scope. Update your application on the [EVE Developers site](#) to ensure it is available.

Add `'allianceauth.corputils'`, to your `INSTALLED_APPS` list in your auth project's settings file. Run migrations to complete installation.

### Creating a Corp Stats

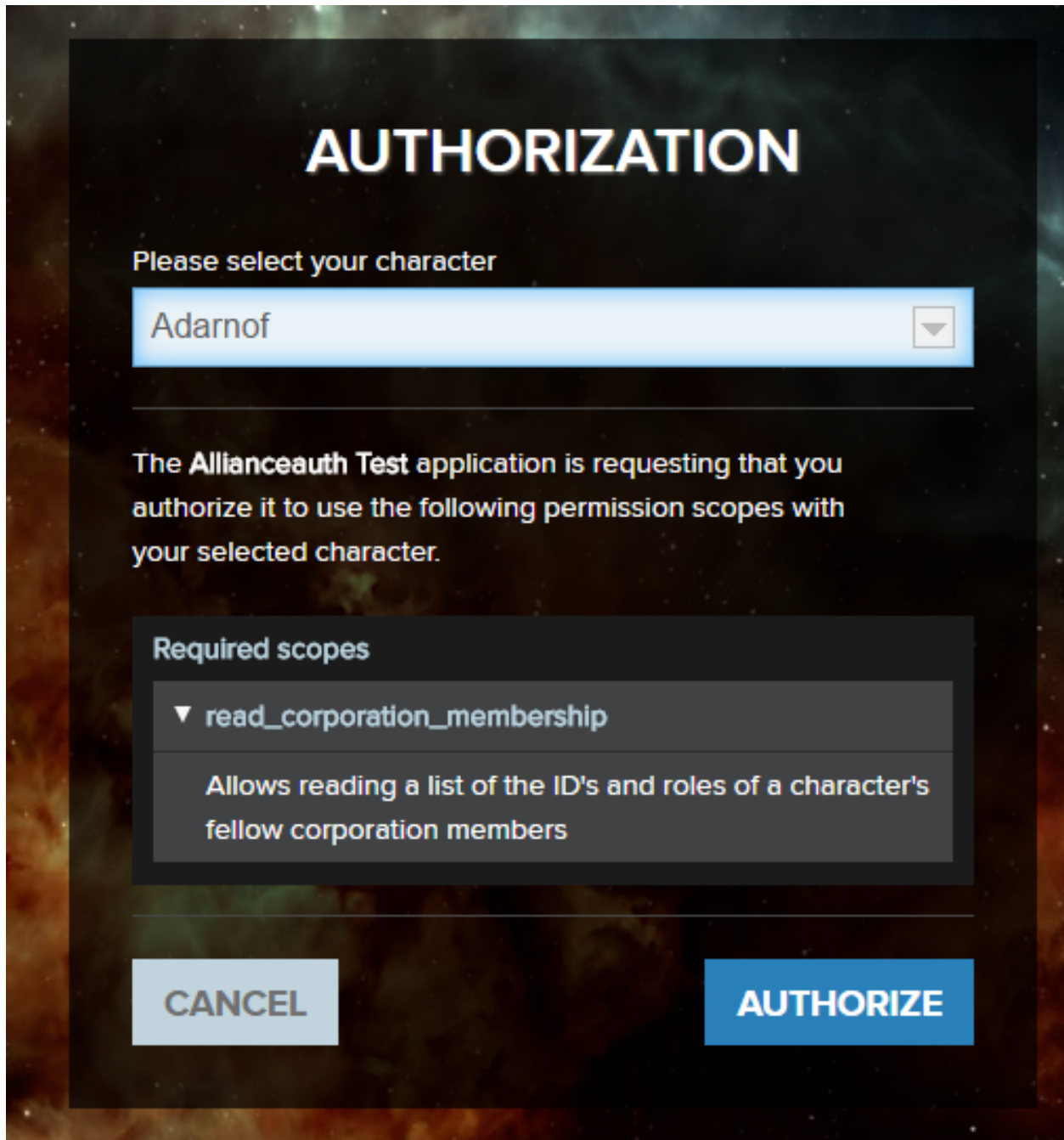
Upon initial install, nothing will be visible. For every Corp, a model will have to be created before data can be viewed.

## Corporation Member Data

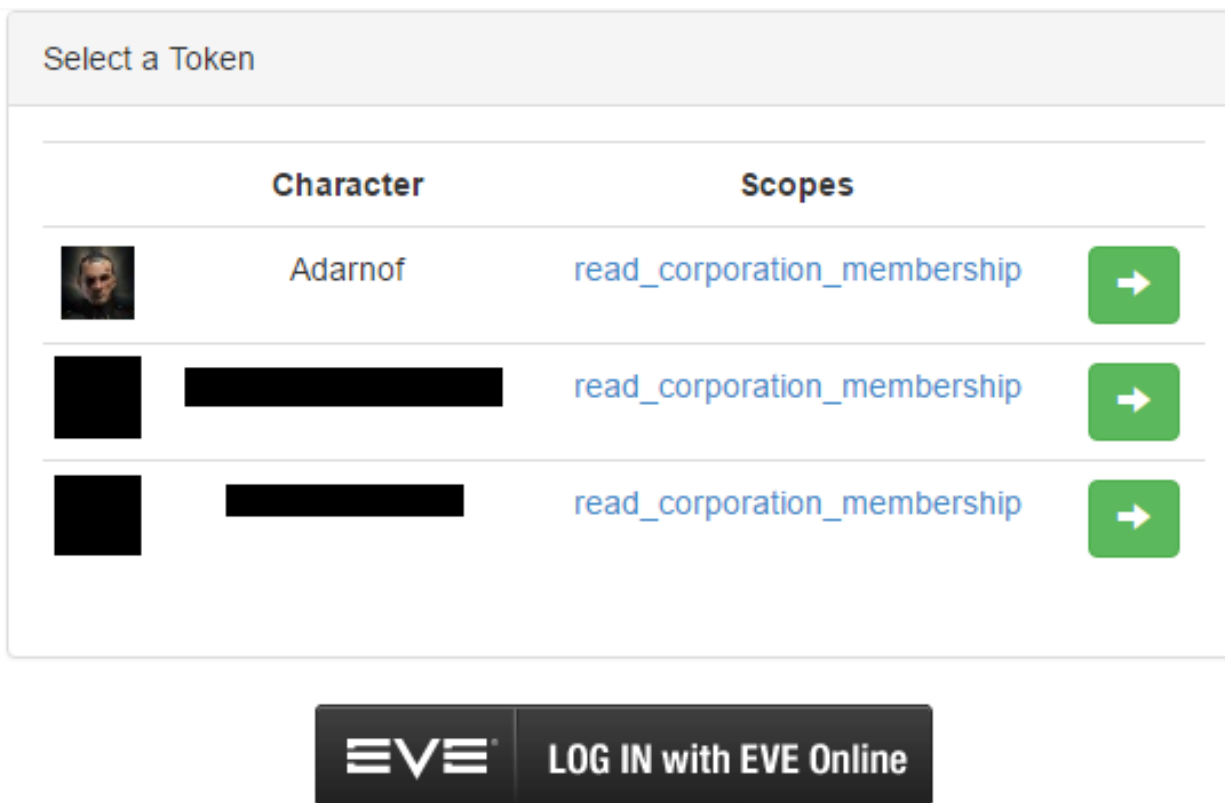


If you are a superuser, the add button will be immediate visible to you. If not, your user account requires the `add_corpstats` permission.

Corp Stats requires an EVE SSO token to access data from the EVE Swagger Interface. Upon pressing the Add button, you will be prompted to authenticated. Please select the character who is in the Corporation you want data for.



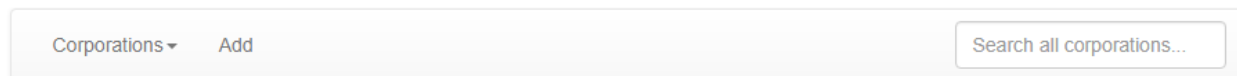
You will return to auth where you are asked to select a token with the green arrow button. If you want to use a different character, press the LOG IN with EVE Online button.



If this works (and you have permission to view the Corp Stats you just created) you'll be returned to a view of the Corp Stats. If it fails an error message will be displayed.

## Corp Stats View

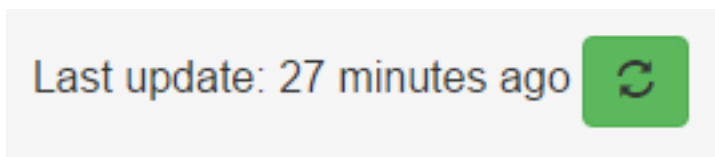
### Navigation Bar



This bar contains a dropdown menu of all available Corporations. If the user has the `add_corpstats` permission, a button to add a Corp Stats will be shown.

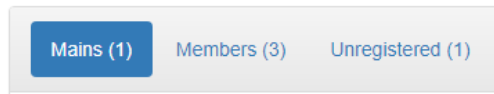
On the right of this bar is a search field. Press enter to search. It checks all characters in all Corp Stats you have view permission to and returns search results.

### Last Update



An update can be performed immediately by pressing the update button. Anyone who can view the Corp Stats can update it.

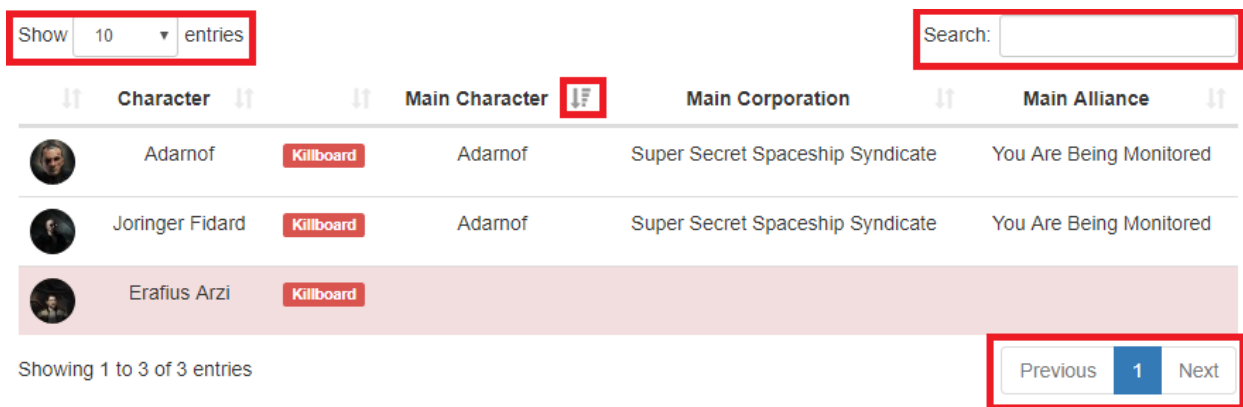
## Character Lists



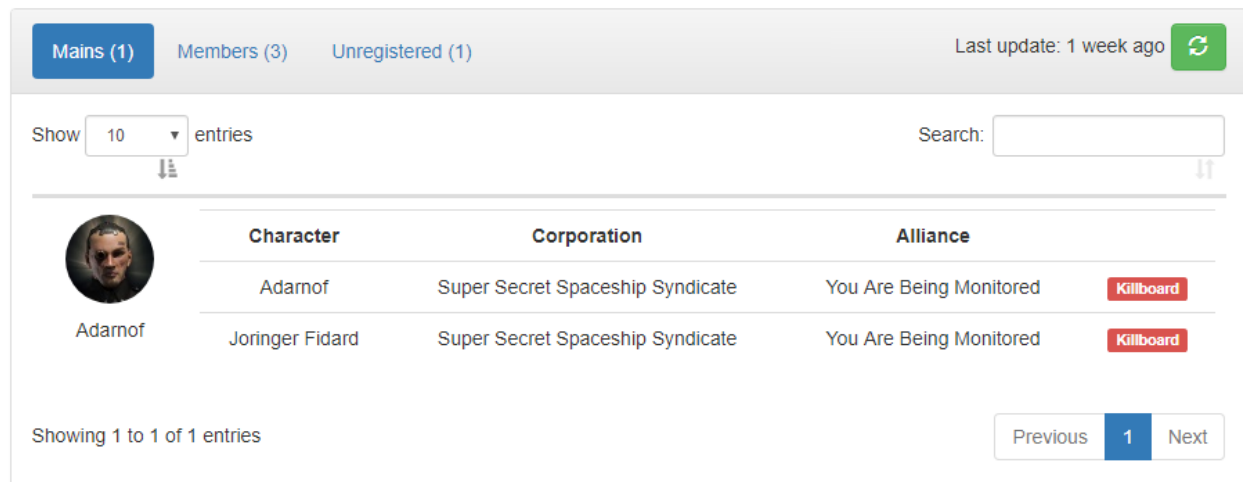
Three views are available:

- main characters and their alts
- registered characters and their main character
- unregistered characters

Each view contains a sortable and searchable table. The number of listings shown can be increased with a dropdown selector. Pages can be changed using the controls on the bottom-right of the table. Each list is searchable at the top-right. Tables can be re-ordered by clicking on column headings.



## Main List



This list contains all main characters in registered in the selected Corporation and their alts. Each character has a link to [zKillboard](#).

## Member List

Mains (1)

Members (3)

Unregistered (1)

Last update: 1 week ago

Show

10

entries

Search:

Character

Main Character

Main Corporation

Main Alliance

Adamof

Killboard

Adamof

Super Secret Spaceship Syndicate

You Are Being Monitored

Erafius Arzi

Killboard

Joringer Fidard

Killboard

Adamof

Super Secret Spaceship Syndicate

You Are Being Monitored

Showing 1 to 3 of 3 entries






Previous

1

Next

The list contains all characters in the Corporation. Red backgrounds means they are not registered in auth. A link to [zKillboard](#) is present for all characters. If registered, the character will also have a main character, main Corporation, and main Alliance field.

## Unregistered List

Mains (1) Members (3) <b>Unregistered (1)</b>			Last update: 1 week ago 		
Show	10	entries	Search: <input type="text"/>		
	Character				
	Erafius Arzi				
Showing 1 to 1 of 1 entries			Previous <b>1</b> Next		

This list contains all characters not registered on auth. Each character has a link to [zKillboard](#).





## Search View

Corporations ▾
Add

Search Results

Show 10 ▾ entries
Search:

Character	Corporation	zKillboard	Main Character	Main Corporation	Main Alliance
 Adarnof	Super Secret Spaceship Syndicate		Adarnof	Super Secret Spaceship Syndicate	You Are Being Monitored

Showing 1 to 1 of 1 entries

Previous
1
Next

This view is essentially the same as the Corp Stats page, but not specific to a single Corporation. The search query is visible in the search box. Characters from all Corp Stats to which the user has view access will be displayed. APIs respect permissions.

## Permissions

To use this feature, users will require some of the following:

Permission	Admin Site	Auth Site
corpstats.view_corp_corpstats	None	Can view corp stats of their corporation.
corpstats.view_alliance_corpstats	None	Can view corp stats of members of their alliance.
corpstats.view_state_corpstats	None	Can view corp stats of members of their auth state.
corpstats.add_corpstats	Can create model	Can add new corpstats using an SSO token.
corpstats.change_corpstats	Can edit model	None.
corpstats.remove_corpstats	Can delete model	None.

Users who add a Corp Stats with their token will be granted permissions to view it regardless of the above permissions. View permissions are interpreted in the “OR” sense: a user can view their Corporations’s Corp Stats without the `view_corp_corpstats` permission if they have the `view_alliance_corpstats` permission, same idea for their state. Note that these evaluate against the user’s main character.

## Automatic Updating

By default Corp Stats are only updated on demand. If you want to automatically refresh on a schedule, add an entry to your project’s settings file:

```
CELERYBEAT_SCHEDULE['update_all_corpstats'] = {
    'task': 'allianceauth.corputils.tasks.update_all_corpstats',
    'schedule': crontab(minute=0, hour="*/6"),
}
```

Adjust the crontab as desired.

## Troubleshooting

### Failure to create Corp Stats

Unrecognized corporation. Please ensure it is a member of the alliance or a blue.

Corp Stats can only be created for Corporations who have a model in the database. These only exist for tenant corps, corps of tenant alliances, blue corps, and members of blue alliances.

Selected corp already has a statistics module.

Only one Corp Stats may exist at a time for a given Corporation.

Failed to gather corporation statistics with selected token.

During initial population, the EVE Swagger Interface did not return any member data. This aborts the creation process. Please wait for the API to start working before attempting to create again.

### Failure to update Corp Stats

Any of the following errors will result in a notification to the owning user, and deletion of the Corp Stats model.

Your token has expired or is no longer valid. Please add a new one to create a new CorpStats.

This occurs when the SSO token is invalid, which can occur when deleted by the user, the character is transferred between accounts, or the API is having a bad day.

CorpStats for (corp name) cannot update with your ESI token as you have left corp.

The SSO token's character is no longer in the Corporation which the Corp Stats is for, and therefore membership data cannot be retrieved.

HTTPForbidden

The SSO token lacks the required scopes to update membership data.

## 3.1.7 Permissions Auditing

Access to most of Alliance Auth's features are controlled by Django's permissions system. In order to help you secure your services, Alliance Auth provides a permissions auditing tool.

### Installation

Add `'allianceauth.permissions_tool'`, to your `INSTALLED_APPS` list in your auth project's settings file.

### Usage

#### Access

In order to grant users access to the permissions auditing tool they will need to be granted the `permissions_tool.audit_permissions` permission or be a superuser.

When a user has access to the tool they will see the "Permissions Audit" menu item under the "Util" sub menu.

## Permissions Overview

The first page gives you a general overview of permissions and how many users have access to each permission.

### Permissions Overview

Showing only applied permissions [Show All](#)

App	Model	Code Name	Name	Users	Groups	Groups Users
discord	discorduser	<a href="#">access_discord</a>	Can access the Discord service	1	2	3
discourse	discourseuser	<a href="#">access_discourse</a>	Can access the Discourse service	0	1	2
groupmanagement	authgroup	<a href="#">request_groups</a>	Can request non-public groups	0	1	2
ipboard	ipboarduser	<a href="#">access_ipboard</a>	Can access the IPBoard service	0	1	2
ips4	ips4user	<a href="#">access_ips4</a>	Can access the IPS4 service	0	1	2
market	marketuser	<a href="#">access_market</a>	Can access the Evernus Market service	0	1	2
mumble	mumbleuser	<a href="#">access_mumble</a>	Can access the Mumble service	0	1	2
openfire	openfireuser	<a href="#">access_openfire</a>	Can access the Openfire service	0	1	2
phpbb3	phpbb3user	<a href="#">access_phpbb3</a>	Can access the phpBB3 service	0	1	2
smf	smfuser	<a href="#">access_smf</a>	Can access the SMF service	0	1	2
teamspeak3	teamspeak3user	<a href="#">access_teamspeak3</a>	Can access the Teamspeak3 service	0	1	2
xenforo	xenforouser	<a href="#">access_xenforo</a>	Can access the XenForo service	0	1	2

**App**, **Model** and **Code Name** contain the internal details of the permission while **Name** contains the name/description you'll see in the admin panel.

**Users** is the number of users explicitly granted this permission on their account.

**Groups** is the number of groups with this permission assigned.

**Groups Users** is the total number of users in all of the groups with this permission assigned.

Clicking on the **Code Name** link will take you to the [Permissions Audit Page](#)

## Permissions Audit Page

The permissions audit page will give you an overview of all the users who have access to this permission either directly or granted via group membership.

## Permissions Audit: access\_discord

<a href="#">&lt; Back</a>	
Group	User
Permission Granted Directly (No Group)	root
Member	basraah
	basraah3
Blue	basraah4

Please note that users may appear multiple times if this permission is granted via multiple sources.

### 3.1.8 Services Name Formats

---

**Note:** New in 2.0

---

Each service's username or nickname, depending on which the service supports, can be customised through the use of the Name Formatter config provided the service supports custom formats. This config can be found in the admin panel under **Services -> Name format config**

Currently the following services support custom name formats:

Service	Used with	Default Formatter
Discord	Nickname	{character_name}
Discourse	Username	{character_name}
IPS4	Username	{character_name}
Mumble	Username	[{corp_ticker}]{character_name}
Openfire	Username	{character_name}
phpBB3	Username	{character_name}
SeAT	Username	{character_name}
SMF	Username	{character_name}
Teamspeak 3	Nickname	[{corp_ticker}]{character_name}
Xenforo	Username	{character_name}

---

**Note:** It's important to note here, before we get into what you can do with a name formatter, that before the generated name is passed off to the service to create an account it will be sanitised to remove characters (the letters and numbers etc.) that the service cannot support. This means that, despite what you configured, the service may display something different. It is up to you to test your formatter and understand how your format may be disrupted by a certain services sanitisation function.

---

#### Available format data

The following fields are available from a users account and main character:

- username - Alliance Auth username
- character\_id
- character\_name
- corp\_id
- corp\_name
- corp\_ticker
- alliance\_id
- alliance\_name
- alliance\_ticker
- alliance\_or\_corp\_name (defaults to Corporation name if there is no Alliance)
- alliance\_or\_corp\_ticker (defaults to Corporation ticker if there is no Alliance)

### Building a formatter string

The name formatter uses the advanced string formatting specified by [PEP-3101](#). Anything supported by this specification is supported in a name formatter.

A more digestible documentation of string formatting in Python is available on the [PyFormat](#) website.

Some examples of strings you could use:

Formatter	Result
{alliance_ticker} -{character_name}	MYALLI -My Character
[{corp_ticker}] {character_name}	[CORP] My Character
{{{corp_name}}}{character_name}	{My Corp}My Character

**Important:** For most services, name formats only take effect when a user creates an account. This means if you create or update a name formatter it won't retroactively alter the format of users names. There are some exceptions to this where the service updates nicknames on a periodic basis. Check the service's documentation to see which of these apply.

**Important:** You must only create one formatter per service per state. E.g. don't create two formatters for Mumble for the Member state. In this case one of the formatters will be used and it may not be the formatter you are expecting.

## 3.1.9 Fleetup

### Installation

Add 'allianceauth.fleetup', to your auth project's INSTALLED\_APPS list.

Additional settings are required. Append the following settings to the end of your auth project's settings file and fill them out.

```
FLEETUP_APP_KEY = '' # The app key from http://fleet-up.com/Api/MyApps
FLEETUP_USER_ID = '' # The user id from http://fleet-up.com/Api/MyKeys
FLEETUP_API_ID = '' # The API id from http://fleet-up.com/Api/MyKeys
FLEETUP_GROUP_ID = '' # The id of the group you want to pull data from, see http://
fleet-up.com/Api/Endpoints#groups_mygroupmemberships
```

---

Once filled out restart Gunicorn and Celery.

### Permissions

The Fleetup module is only visible to users with the `auth | user | view_fleeup` permission.

### 3.1.10 Fleet Activity Tracking

#### Installation

Fleet Activity Tracking requires access to the `esi-location.read_location.v1`, `esi-location.read_ship_type.v1`, and `esi-universe.read_structures.v1` SSO scopes. Update your application on the [EVE Developers site](#) to ensure these are available.

Add `'allianceauth.fleetactivitytracking'`, to your `INSTALLED_APPS` list in your auth project's settings file. Run migrations to complete installation.

### 3.1.11 Optimer

#### Installation

Add `'allianceauth.optimer'`, to your `INSTALLED_APPS` list in your auth project's settings file. Run migrations to complete installation.

### 3.1.12 SRP

#### Installation

Add `'allianceauth.srp'`, to your `INSTALLED_APPS` list in your auth project's settings file. Run migrations to complete installation.

### 3.1.13 Timerboard

#### Installation

Add `'allianceauth.timerboard'`, to your `INSTALLED_APPS` list in your auth project's settings file. Run migrations to complete installation.

## 3.2 Installation

### 3.2.1 Auth

#### Alliance Auth Installation

---

**Tip:** If you are uncomfortable with Linux permissions follow the steps below as the root user.

---

## Dependencies

Alliance Auth can be installed on any operating system. Dependencies are provided below for two of the most popular server platforms, Ubuntu and CentOS. To install on your favourite flavour of Linux, identify and install equivalent packages to the ones listed here.

---

**Hint:** CentOS: A few packages are included in a non-default repository. Add it and update the package lists.

```
yum -y install https://centos7.iuscommunity.org/ius-release.rpm
yum update
```

---

## Python

Alliance Auth requires python3.4 or higher. Ensure it is installed on your server before proceeding.

Ubuntu:

```
apt-get install python3 python3-dev python3-venv python3-setuptools python3-pip
```

CentOS:

```
yum install python36u python36u-devel python36u-setuptools python36u-pip
```

## Database

It's recommended to use a database service instead of SQLite. Many options are available, but this guide will use MariaDB.

Ubuntu:

```
apt-get install mariadb-server mariadb-client libmysqlclient-dev
```

CentOS:

```
yum install mariadb-server mariadb-devel mariadb-shared mariadb
```

---

**Note:** If you don't plan on running the database on the same server as auth you still need to install the libmysqlclient-dev package on Ubuntu or mariadb-devel package on CentOS.

---

## Redis and Other Tools

A few extra utilities are also required for installation of packages.

Ubuntu:

```
apt-get install unzip git redis-server curl libssl-dev libbz2-dev libffi-dev
```

CentOS:

```
yum install gcc gcc-c++ unzip git redis curl bzip2-devel
```

---

**Important:** CentOS: Make sure Redis is running before continuing.

```
systemctl enable redis.service  
systemctl start redis.service
```

---

## Database Setup

Alliance Auth needs a MySQL user account and database. Open an SQL shell with `mysql -u root -p` and create them as follows, replacing `PASSWORD` with an actual secure password:

```
CREATE USER 'allianceserver'@'localhost' IDENTIFIED BY 'PASSWORD';  
CREATE DATABASE alliance_auth CHARACTER SET utf8mb4;  
GRANT ALL PRIVILEGES ON alliance_auth . * TO 'allianceserver'@'localhost';
```

Add timezone tables to your mysql installation:

```
mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root -p mysql
```

---

**Note:** You may see errors when you add the timezone tables. To make sure that they were correctly added run the following commands and check for the `time_zone` tables:

```
mysql -u root -p  
use mysql;  
show tables;
```

---

Close the SQL shell and secure your database server with the `mysql_secure_installation` command.

## Auth Install

### User Account

For security and permissions, it's highly recommended you create a separate user to install auth under. Do not log in as this account.

Ubuntu:

```
adduser --disabled-login allianceserver
```

CentOS:

```
useradd -s /bin/nologin allianceserver
```



## Virtual Environment

Create a Python virtual environment and put it somewhere convenient (e.g. `/home/allianceserver/venv/auth/`)

```
python3 -m venv /home/allianceserver/venv/auth/
```

**Warning:** The `python3` command may not be available on all installations. Try a specific version such as `python3.6` if this is the case.

**Tip:** A virtual environment provides support for creating a lightweight “copy” of Python with their own site directories. Each virtual environment has its own Python binary (allowing creation of environments with various Python versions) and can have its own independent set of installed Python packages in its site directories. You can read more about virtual environments on the [Python docs](#).

Activate the virtualenv using `source /home/allianceserver/venv/auth/bin/activate`. Note the `/bin/activate` on the end of the path.

**Hint:** Each time you come to do maintenance on your Alliance Auth installation, you should activate your virtual environment first. When finished, deactivate it with the `deactivate` command.

Ensure wheel is available with `pip install wheel` before continuing.

## Alliance Auth Project

You can install the library using `pip install allianceauth`. This will install Alliance Auth and all its python dependencies. You should also install Gunicorn with `pip install gunicorn` before proceeding.

Now you need to create the application that will run the Alliance Auth install. Ensure you are in the `allianceserver` home directory by issuing `cd /home/allianceserver`.

The `allianceauth start myauth` command bootstraps a Django project which will run Alliance Auth. You can rename it from `myauth` to anything you’d like: this name is shown by default as the site name but that can be changed later.

The settings file needs configuring. Edit the template at `myauth/myauth/settings/local.py`. Be sure to configure the EVE SSO and Email settings.

Django needs to install models to the database before it can start.

```
python /home/allianceserver/myauth/manage.py migrate
```

Now we need to round up all the static files required to render templates. Make a directory to serve them from and populate it.

```
mkdir -p /var/www/myauth/static
python /home/allianceserver/myauth/manage.py collectstatic
```

Check to ensure your settings are valid.

```
python /home/allianceserver/myauth/manage.py check
```

And finally ensure the allianceserver user has read/write permissions to this directory before proceeding.

```
chown -R allianceserver:allianceserver /home/allianceserver/myauth
```

## Background Tasks

### Gunicorn

To run the auth website a [WSGI Server](#) is required. [Gunicorn](#) is highly recommended for its ease of configuring. It can be manually run from within your myauth base directory with `gunicorn --bind 0.0.0.0 myauth.wsgi` or automatically run using Supervisor.

The default configuration is good enough for most installations. Additional information is available in the [gunicorn](#) doc.

### Supervisor

[Supervisor](#) is a process watchdog service: it makes sure other processes are started automatically and kept running. It can be used to automatically start the WSGI server and Celery workers for background tasks. Installation varies by OS:

Ubuntu:

```
apt-get install supervisor
```

CentOS:

```
yum install supervisor
systemctl enable supervisord.service
systemctl start supervisord.service
```

Once installed it needs a configuration file to know which processes to watch. Your Alliance Auth project comes with a ready-to-use template which will ensure the Celery workers, Celery task scheduler and Gunicorn are all running.

Ubuntu:

```
ln -s /home/allianceserver/myauth/supervisor.conf /etc/supervisor/conf.d/myauth.conf
```

CentOS:

```
ln -s /home/allianceserver/myauth/supervisor.conf /etc/supervisord.d/myauth.ini
```

And activate it with `supervisorctl reload`.

You can check the status of the processes with `supervisorctl status`. Logs from these processes are available in `/home/allianceserver/myauth/log` named by process.

---

**Note:** Any time the code or your settings change you'll need to restart Gunicorn and Celery.

```
supervisorctl restart myauth:
```

---

## Webserver

Once installed, decide on whether you're going to use *NGINX* or *Apache* and follow the respective guide.

## Superuser

Before using your auth site it is essential to create a superuser account. This account will have all permissions in Alliance Auth. It's OK to use this as your personal auth account.

```
python /home/allianceserver/myauth/manage.py createsuperuser
```

The superuser account is accessed by logging in via the admin site at `https://example.com/admin`.

If you intend to use this account as your personal auth account you need to add a main character. Navigate to the normal user dashboard (at `https://example.com`) after logging in via the admin site and select `Change Main`. Once a main character has been added it is possible to use SSO to login to this account.

## Updating

Periodically [new releases](#) are issued with bug fixes and new features. To update your install, simply activate your virtual environment and update with `pip install --upgrade allianceauth`. Be sure to read the release notes which will highlight changes.

Some releases come with changes to settings: update your project's settings with `allianceauth update /home/allianceserver/myauth`.

Some releases come with new or changed models. Update your database to reflect this with `python /home/allianceserver/myauth/manage.py migrate`.

Always restart Celery and Gunicorn after updating.

## Upgrading from v1.15

It's possible to preserve a v1 install's database and migrate it to v2. This will retain all service accounts, user accounts with their main character, but will purge API keys and alts.

## Preparing to Upgrade

Ensure your auth install is at least version 1.15 - it's preferred to be on v1.15.8 but any v1.15.x should work. If not, update following normal procedures and ensure you run migrations.

If you will not be using any apps (or they have been removed in v2) clear their database tables before beginning the v2 install procedure. From within your v1 install, `python manage.py migrate APPNAME zero`, replacing APPNAME with the appropriate name.

It's strongly encouraged to perform the upgrade with a **copy** of the original v1 database. If something goes wrong this allows you to roll back and try again. This can be achieved with:

```
mysqldump -u root -p v1_database_name_here | mysql -u root -p v2_database_name_here
```

Note this command will prompt you for the root password twice.

Alliance Auth v2 requires Python 3.4 or newer. If this is not available on your system be sure to install the dependencies listed.

### Installation

Follow the *normal install procedures for Alliance Auth v2*. If you're coming from v1 you likely already have the user account and dependencies.

When configuring settings, ensure to enter the database information relating to the copy you made of the v1 database, but **do not run migrations**. See below before continuing.

Do not start Celery yet.

### Migration

During the upgrade process a migration is run which ports data to the new system. Its behaviour can be configured through some optional settings.

### User Accounts

A database migration is included to port users to the new SSO-based login system. It will automatically assign states and main characters.

Password for non-staff accounts are destroyed so they cannot be used to log in - only SSO is available for regular members. Staff can login using username and password through the admin site or by SSO.

### EVE Characters

Character ownership is now tracked by periodically validating a refreshable SSO token. When migrating from v1 not all users may have such a refreshable token: to allow these users to retain their existing user account their main character is set to the one present in v1, bypassing this validation mechanism.

It is essential to get your users to log in via SSO soon after migration to ensure their accounts are managed properly. Any user who does not log in will not lose their main character under any circumstance, even character sale. During a sale characters are transferred to an NPC corp - get celery tasks running within 24 hours of migration so that during a sale the user loses their state (and hence permissions). This can be an issue if you've granted service access permissions to a user or their groups: it's strongly encouraged to grant service access by state from now on.

Because character ownership is tracked separately of main character it is not possible to preserve alts unless a refreshable SSO token is present for them.

### Members and Blues

The new *state system* allows configuring dynamic membership states through the admin page. Unfortunately if you make a change after migrating it will immediately assess user states and see that no one should be a member. You can add additional settings to your auth project's settings file to generate the member and blue states as you have them defined in v1:

- `ALLIANCE_IDS = []` a list of member alliance IDs
- `CORP_IDS = []` a list of member corporation IDs
- `BLUE_ALLIANCE_IDS = []` a list of blue alliance IDs
- `BLUE_CORP_IDS = []` a list of blue corporation IDs

Put comma-separated IDs into the brackets and the migration will create states with the members and blues you had before. This will prevent unexpected state purging when you edit states via the admin site.

## Default Groups

If you used member/blue group names other than the standard “Member” and “Blue” you can enter settings to have the member/blue states created through this migration take these names.

- `DEFAULT_AUTH_GROUP = ""` the desired name of the “Member” state
- `DEFAULT_BLUE_GROUP = ""` the desired name of the “Blue” state

Any permissions assigned to these groups will be copied to the state replacing them. Because these groups are no longer managed they pose a security risk and so are deleted at the end of the migration automatically.

## Run Migrations

Once you’ve configured any optional settings it is now safe to run the included migrations.

## Validating Upgrade

Before starting the Celery workers it’s a good idea to validate the states were created and assigned correctly. Any mistakes now will trigger deletion of service accounts: if Celery workers aren’t running these tasks aren’t yet processed. States can be checked through the admin site.

The site (and not Celery) can be started with `supervisorctl start myauth:gunicorn`. Then navigate to the admin site and log in with your v1 username and password. States and User Profiles can be found under the Authentication app.

Once you have confirmed states are correctly configured (or adjusted them as needed) clear the Celery task queue: from within your auth project folder, `celery -A myauth purge`

It should now be safe to bring workers online (`supervisorctl start myauth:`) and invite users to use the site.

## Securing User Accounts

After migration users will have main characters without a method to check for character sales. After a brief period to allow your users to log in (and provide a refreshable token to use), it’s a good idea to clear main characters of users who haven’t yet logged in. This can be achieved with an included command: `python manage.py checkmains`

A similar process can be used to ensure users who may have lost service permissions during the migration do not have active service accounts. This is done using `python manage.py validate_service_accounts`.

## Help

If something goes wrong during the migration reach out for help on [Gitter](#) or open an [issue](#).

## Gunicorn

[Gunicorn](#) is a Python WSGI HTTP Server for UNIX. The Gunicorn server is light on server resources, and fairly speedy.

If you find Apache’s `mod_wsgi` to be a headache or want to use NGINX (or some other webserver), then Gunicorn could be for you. There are a number of other WSGI server options out there and this documentation should be enough for you to piece together how to get them working with your environment.

Check out the full [Gunicorn docs](#).

### Setting up Gunicorn

---

**Note:** If you're using a virtual environment, activate it now. `source /path/to/venv/bin/activate`.

---

Install Gunicorn using `pip`, `pip install gunicorn`.

In your `myauth` base directory, try running `gunicorn --bind 0.0.0.0:8000 myauth.wsgi`. You should be able to browse to `http://yourserver:8000` and see your Alliance Auth installation running. Images and styling will be missing, but don't worry, your web server will provide them.

Once you validate its running, you can kill the process with `Ctrl+C` and continue.

### Running Gunicorn with Supervisor

You should use Supervisor to keep all of Alliance Auth components running (instead of using `screen`). You don't *have to* but we will be using it to start and run Gunicorn so you might as well.

### Sample Supervisor config

You'll want to edit `/etc/supervisor/conf.d/myauth_gunicorn.conf` (or whatever you want to call the config file)

```
[program:myauth-gunicorn]
user = allianceserver
directory=/home/allianceserver/myauth/
command=gunicorn myauth.wsgi --workers=3 --timeout 120
autostart=true
autorestart=true
stopsignal=INT
```

- `[program:myauth-gunicorn]` - Change `myauth-gunicorn` to whatever you wish to call your process in Supervisor.
- `user = allianceserver` - Change to whatever user you wish Gunicorn to run as. You could even set this as `allianceserver` if you wished. I'll leave the question security of that up to you.
- `directory=/home/allianceserver/myauth/` - Needs to be the path to your Alliance Auth project.
- `command=gunicorn myauth.wsgi --workers=3 --timeout 120` - Running Gunicorn and the options to launch with. This is where you have some decisions to make, we'll continue below.

### Gunicorn Arguments

See the [Commonly Used Arguments](#) or [Full list of settings](#) for more information.

## Where to bind Gunicorn to?

What address are you going to use to reference it? By default, without a bind parameter, Gunicorn will bind to `127.0.0.1:8000`. This might be fine for your application. If it clashes with another application running on that port you will need to change it. I would suggest using UNIX sockets too, if you can.

For UNIX sockets add `--bind=unix:/run/allianceauth.sock` (or to a path you wish to use). Remember that your web server will need to be able to access this socket file.

For a TCP address add `--bind=127.0.0.1:8001` (or to the address/port you wish to use, but I would strongly advise against binding it to an external address).

Whatever you decide to use, remember it because we'll need it when configuring your webserver.

## Number of workers

By default Gunicorn will spawn only one worker. The number you set this to will depend on your own server environment, how many visitors you have etc. Gunicorn suggests between 2-4 workers per core. Really you could probably get away with 2-4 in total for most installs.

Change it by adding `--workers=2` to the command.

## Running with a virtual environment

If you're running with a virtual environment, you'll need to add the path to the `command=` config line.

e.g. `command=/path/to/venv/bin/gunicorn myauth.wsgi`

## Starting via Supervisor

Once you have your configuration all sorted, you will need to reload your supervisor config `service supervisor reload` and then you can start the Gunicorn server via `supervisorctl start aauth-gunicorn` (or whatever you renamed it to). You should see something like the following `aauth-gunicorn: started`. If you get some other message, you'll need to consult the Supervisor log files, usually found in `/var/log/supervisor/`.

## Configuring your webserver

Any web server capable of proxy passing should be able to sit in front of Gunicorn. Consult their documentation armed with your `--bind=` address and you should be able to find how to do it relatively easy.

## Restarting Gunicorn

In the past when you made changes you restarted the entire Apache server. This is no longer required. When you update or make configuration changes that ask you to restart Apache, instead you can just restart Gunicorn:

`supervisorctl restart myauth-gunicorn`, or the service name you chose for it.

## NGINX

### Overview

Nginx (engine x) is a HTTP server known for its high performance, stability, simple configuration, and low resource consumption. Unlike traditional servers (i.e. Apache), Nginx doesn't rely on threads to serve requests, rather using an asynchronous event driven approach which permits predictable resource usage and performance under load.

If you're trying to cram Alliance Auth into a very small VPS of say, 1-2GB or less, then Nginx will be considerably friendlier to your resources compared to Apache.

You can read more about NGINX on the [NGINX wiki](#).

### Coming from Apache

If you're converting from Apache, here are some things to consider.

Nginx is lightweight for a reason. It doesn't try to do everything internally and instead concentrates on just being a good HTTP server. This means that, unlike Apache, it won't automatically run PHP scripts via `mod_php` and doesn't have an internal WSGI server like `mod_wsgi`. That doesn't mean that it can't, just that it relies on external processes to run these instead. This might be good or bad depending on your outlook. It's good because it allows you to segment your applications, restarting Alliance Auth won't impact your PHP applications. On the other hand it means more config and more management of services. For some people it will be worth it, for others losing the centralised nature of Apache may not be worth it.

Apache	Nginx Replacement
<code>mod_php</code>	<code>php5-fpm</code> or <code>php7-fpm</code> (PHP FastCGI)
<code>mod_wsgi</code>	Gunicorn or other external WSGI server

Your `.htaccess` files won't work. Nginx has a separate way of managing access to folders via the server config. Everything you can do with `htaccess` files you can do with Nginx config. [Read more on the Nginx wiki](#)

### Setting up Nginx

Install Nginx via your preferred package manager or other method. If you need help just search, there are plenty of guides on installing Nginx out there.

Nginx needs to be able to read the folder containing your auth project's static files. `chown -R nginx:nginx /var/www/myauth/static`.

---

**Tip:** Some specific distros may use `www-data:www-data` instead of `nginx:nginx`, causing static files (images, stylesheets etc) not to appear. You can confirm what user Nginx will run under by checking either its base config file `/etc/nginx/nginx.conf` for the "user" setting, or once Nginx has started `ps aux | grep nginx`. Adjust your `chown` commands to the correct user if needed.

---

You will need to have [Gunicorn](#) or some other WSGI server setup for hosting Alliance Auth.

### Ubuntu

Create a config file in `/etc/nginx/sites-available` and call it `alliance-auth.conf` or whatever your preferred name is.



Create a symbolic link to enable the site `ln -s /etc/nginx/sites-available/alliance-auth.conf /etc/nginx/sites-enabled/`

## CentOS

Create a config file in `/etc/nginx/conf.d` and call it `alliance-auth.conf` or whatever your preferred name is.

## Basic config

Copy this basic config into your config file. Make whatever changes you feel are necessary.

```
server {
    listen 80;
    server_name example.com;

    location = /favicon.ico { access_log off; log_not_found off; }

    location /static {
        alias /var/www/myauth/static;
        autoindex off;
    }

    # Unicorn config goes below
    location / {
        include proxy_params;
        proxy_pass http://127.0.0.1:8000;
    }
}
```

Restart Nginx after making changes to the config files. On Ubuntu `service nginx restart` and on CentOS `systemctl restart nginx.service`.

## Adding TLS/SSL

With [Let's Encrypt](#) offering free SSL certificates, there's no good reason to not run HTTPS anymore. The bot can automatically configure Nginx on some operating systems. If not proceed with the manual steps below.

Your config will need a few additions once you've got your certificate.

```
listen 443 ssl http2; # Replace listen 80; with this

ssl_certificate      /path/to/your/cert.crt;
ssl_certificate_key  /path/to/your/cert.key;

ssl on;
ssl_session_cache    builtin:1000 shared:SSL:10m;
ssl_protocols        TLSv1 TLSv1.1 TLSv1.2;
ssl_ciphers           EECDH+ECDSA+AESGCM:EECDH+aRSA+AESGCM:EECDH+ECDSA+SHA384:
↪EECDH+ECDSA+SHA256:EECDH+aRSA+SHA384:EECDH+aRSA+SHA256:EECDH+aRSA+RC4:EECDH:
↪EDH+aRSA:RC4:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PSK:!SRP:!DSS;
ssl_prefer_server_ciphers on;
```

If you want to redirect all your non-SSL visitors to your secure site, below your main configs `server` block, add the following:

```
server {
    listen 80;
    server_name example.com;

    # Redirect all HTTP requests to HTTPS with a 301 Moved Permanently response.
    return 301 https://$host$request_uri;
}
```

If you have trouble with the `ssl_ciphers` listed here or some other part of the SSL config, try getting the values from [Mozilla's SSL Config Generator](#).

## Apache

### Overview

Alliance Auth gets served using a Web Server Gateway Interface (WSGI) script. This script passes web requests to Alliance Auth which generates the content to be displayed and returns it. This means very little has to be configured in Apache to host Alliance Auth.

If you're using a small VPS to host services with very limited memory, consider using [NGINX](#).

### Installation

Ubuntu:

```
apt-get install apache2
```

CentOS:

```
yum install httpd
systemctl enable httpd
systemctl start httpd
```

### Configuration

Apache needs to be able to read the folder containing your auth project's static files. On Ubuntu: `chown -R www-data:www-data /var/www/myauth/static`, and on CentOS: `chown -R apache:apache /var/www/myauth/static`

Apache serves sites through defined virtual hosts. These are located in `/etc/apache2/sites-available/` on Ubuntu and `/etc/httpd/conf.d/httpd.conf` on CentOS.

A virtual host for auth need only proxy requests to your WSGI server (Gunicorn if you followed the install guide) and serve static files. Examples can be found below. Create your config in its own file e.g. `myauth.conf`

### Ubuntu

To proxy and modify headers a few mods need to be enabled.

```
a2enmod proxy
a2enmod proxy_http
a2enmod headers
```

Create a new config file for auth e.g. `/etc/apache2/sites-available/myauth.conf` and fill out the virtual host configuration. To enable your config use `a2ensite myauth.conf` and then reload apache with `service apache2 reload`.

## CentOS

Place your virtual host configuration in the appropriate section within `/etc/httpd/conf.d/httpd.conf` and restart the httpd service with `systemctl restart httpd`.

## Sample Config File

```
<VirtualHost *:80>
    ServerName auth.example.com

    ProxyPassMatch ^/static !
    ProxyPass / http://127.0.0.1:8000/
    ProxyPassReverse / http://127.0.0.1:8000/
    ProxyPreserveHost On

    Alias "/static" "/var/www/myauth/static"
    <Directory "/var/www/myauth/static">
        Require all granted
    </Directory>
</VirtualHost>
```

## SSL

It's 2018 - there's no reason to run a site without SSL. The EFF provides free, renewable SSL certificates with an automated installer. Visit their [website](#) for information.

After acquiring SSL the config file needs to be adjusted. Add the following lines inside the `<VirtualHost>` block:

```
RequestHeader set X-FORWARDED-PROTOCOL https
RequestHeader set X-FORWARDED-SSL On
```

## 3.2.2 Services

### Service Permissions

In the past, access to services was dictated by a list of settings in `settings.py`, granting access to each particular service for Members and/or Blues. This meant that granting access to a service was very broad and rigidly structured around these two states.

### Permissions based access

Instead of granting access to services by the previous rigid structure, access to services is now granted by the built in Django permissions system. This means that service access can be more granular, allowing only certain states, certain groups, for instance Corp CEOs, or even individual user access to each enabled service.

---

**Important:** If you grant access to an individual user, they will have access to that service regardless of whether or not they are a member.

---

Each service has an access permission defined, named like `Can access the <service name> service`.

To mimick the old behaviour of enabling services for all members, you would select the `Member` group from the admin panel, add the required service permission to the group and save. Likewise for Blues, select the `Blue` group and add the required permission.

A user can be granted the same permission from multiple sources. e.g. they may have it granted by several groups and directly granted on their account as well. Auth will not remove their account until all instances of the permission for that service have been revoked.

### Removing access

**Danger:** Access removal is processed immediately after removing a permission from a user or group. If you remove access from a large group, such as `Member`, it will immediately remove all users from that service.

When you remove a service permission from a user, a signal is triggered which will activate an immediate permission check. For users this will trigger an access check for all services. For groups, due to the potential extra load, only the services whose permissions have changed will be verified, and only the users in that group.

If a user no longer has permission to access the service when this permissions check is triggered, that service will be immediately disabled for them.

### Disabling user accounts

When you unset a user as active in the admin panel, all of that users service accounts will be immediately disabled or removed. This is due to the built in behaviour of the Django permissions system, which will return `False` for all permissions if a users account is disabled, regardless of their actual permissions state.

## Alliance Market

### Deprecation

Alliance Market relies on the now non-functional XML API.

Please remove this service data with `python manage.py migrate appname zero` and then remove from your `INSTALLED_APPS` list.

## Discord

### Overview

Discord is a web-based instant messaging client with voice. Kind of like TeamSpeak meets Slack meets Skype. It also has a standalone app for phones and desktop.

Discord is very popular amongst ad-hoc small groups and larger organizations seeking a modern technology. Alternative voice communications should be investigated for larger than small-medium groups for more advanced features.

### Setup

#### Prepare Your Settings File

In your auth project's settings file, do the following:

- Add `'allianceauth.services.modules.discord'`, to your `INSTALLED_APPS` list
- Append the following to the bottom of the settings file:

```
# Discord Configuration
DISCORD_GUILD_ID = ''
DISCORD_CALLBACK_URL = ''
DISCORD_APP_ID = ''
DISCORD_APP_SECRET = ''
DISCORD_BOT_TOKEN = ''
DISCORD_SYNC_NAMES = False
```

#### Creating a Server

Navigate to the [Discord site](#) and register an account, or log in if you have one already.

On the left side of the screen you'll see a circle with a plus sign. This is the button to create a new server. Go ahead and do that, naming it something obvious.

Now retrieve the server ID [following this procedure](#).

Update your auth project's settings file, inputting the server ID as `DISCORD_GUILD_ID`

---

**Note:** If you already have a Discord server skip the creation step, but be sure to retrieve the server ID

---

#### Registering an Application

Navigate to the [Discord Developers site](#). Press the plus sign to create a new application.

Give it a name and description relating to your auth site. Add a redirect to `https://example.com/discord/callback/`, substituting your domain. Press Create Application.

Update your auth project's settings file, inputting this redirect address as `DISCORD_CALLBACK_URL`

On the application summary page, press Create a Bot User.

Update your auth project's settings file with these pieces of information from the summary page:

- From the App Details panel, `DISCORD_APP_ID` is the Client/Application ID

- From the App Details panel, `DISCORD_APP_SECRET` is the Secret
- From the App Bot Users panel, `DISCORD_BOT_TOKEN` is the Token

### Preparing Auth

Before continuing it is essential to run migrations and restart Gunicorn and Celery.

### Adding a Bot to the Server

Once created, navigate to the services page of your Alliance Auth install as the superuser account. At the top there is a big green button labelled Link Discord Server. Click it, then from the drop down select the server you created, and then Authorize.

This adds a new user to your Discord server with a `BOT` tag, and a new role with the same name as your Discord application. Don't touch either of these. If for some reason the bot loses permissions or is removed from the server, click this button again.

To manage roles, this bot role must be at the top of the hierarchy. Edit your Discord server, roles, and click and drag the role with the same name as your application to the top of the list. This role must stay at the top of the list for the bot to work. Finally, the owner of the bot account must enable 2 Factor Authentication (this is required from Discord for kicking and modifying member roles). If you are unsure what 2FA is or how to set it up, refer to [this support page](#). It is also recommended to force 2FA on your server (this forces any admins or moderators to have 2fa enabled to perform similar functions on discord).

Note that the bot will never appear online as it does not participate in chat channels.

### Linking Accounts

Instead of the usual account creation procedure, for Discord to work we need to link accounts to Alliance Auth. When attempting to enable the Discord service, users are redirected to the official Discord site to authenticate. They will need to create an account if they don't have one prior to continuing. Upon authorization, users are redirected back to Alliance Auth with an OAuth code which is used to join the Discord server.

### Syncing Nicknames

If you want users to have their Discord nickname changed to their in-game character name, set `DISCORD_SYNC_NAMES` to `True`

### Managing Roles

Once users link their accounts you'll notice Roles get populated on Discord. These are the equivalent to Groups on every other service. The default permissions should be enough for members to use text and audio communications. Add more permissions to the roles as desired through the server management window.

### Troubleshooting

## “Unknown Error” on Discord site when activating service

This indicates your callback URL doesn't match. Ensure the `DISCORD_CALLBACK_URL` setting exactly matches the URL entered on the Discord developers site. This includes `http(s)`, trailing slash, etc.

## Discourse

### Prepare Your Settings

In your auth project's settings file, do the following:

- Add `'allianceauth.services.modules.discourse'`, to your `INSTALLED_APPS` list
- Append the following to your `local.py` settings file:

```
# Discourse Configuration
DISCOURSE_URL = ''
DISCOURSE_API_USERNAME = ''
DISCOURSE_API_KEY = ''
DISCOURSE_SSO_SECRET = ''
```

## Install Docker

```
wget -qO- https://get.docker.io/ | sh
```

## Install Discourse

### Download Discourse

```
mkdir /var/discourse
git clone https://github.com/discourse/discourse_docker.git /var/discourse
```

## Configure

```
cd /var/discourse
cp samples/standalone.yml containers/app.yml
nano containers/app.yml
```

Change the following:

- `DISCOURSE_DEVELOPER_EMAILS` should be a list of admin account email addresses separated by commas.
- `DISCOURSE_HOSTNAME` should be `discourse.example.com` or something similar.
- Everything with SMTP depends on your mail settings. [There are plenty of free email services online recommended by Discourse](#) if you haven't set one up for auth already.

To install behind Apache/Nginx, look for this section:

```
...  
## which TCP/IP ports should this container expose?  
expose:  
  - "80:80"    # fwd host port 80    to container port 80 (http)  
...
```

Change it to this:

```
...  
## which TCP/IP ports should this container expose?  
expose:  
  - "7890:80"   # fwd host port 7890  to container port 80 (http)  
...
```

Or any other port will do, if taken. Remember this number.

### Build and launch

```
nano /etc/default/docker
```

Uncomment this line:

```
DOCKER_OPTS="--dns 8.8.8.8 --dns 8.8.4.4"
```

Restart Docker:

```
service docker restart
```

Now build:

```
./launcher bootstrap app  
./launcher start app
```

### Web Server Configuration

You will need to configure your web server to proxy requests to Discourse.

A minimal Apache config might look like:

```
<VirtualHost *:80>  
  ServerName discourse.example.com  
  ProxyPass / http://0.0.0.0:7890/  
  ProxyPassReverse / http://0.0.0.0:7890/  
</VirtualHost>
```

A minimal Nginx config might look like:

```
server {  
  listen 80;  
  server_name discourse.example.com;  
  location / {  
    include proxy_params;  
    proxy_pass http://127.0.0.1:7890;  
  }  
}
```



## Configure API

### Generate admin account

From the `/var/discourse` directory,

```
./launcher enter app
rake admin:create
```

Follow prompts, being sure to answer `y` when asked to allow admin privileges.

### Create API key

Navigate to `discourse.example.com` and log on. Top right press the 3 lines and select Admin. Go to API tab and press Generate Master API Key.

Add the following values to your auth project's settings file:

- `DISCOURSE_URL`: `https://discourse.example.com` (do not add a trailing slash!)
- `DISCOURSE_API_USERNAME`: the username of the admin account you generated the API key with
- `DISCOURSE_API_KEY`: the key you just generated

## Configure SSO

Navigate to `discourse.example.com` and log in. Back to the admin site, scroll down to find SSO settings and set the following:

- `enable_sso`: `True`
- `sso_url`: `http://example.com/discourse/sso`
- `sso_secret`: some secure key

Save, now set `DISCOURSE_SSO_SECRET` in your auth project's settings file to the secure key you just put in Discourse.

Finally run migrations and restart Gunicorn and Celery.

## Mumble

### Prepare Your Settings

In your auth project's settings file, do the following:

- Add `'allianceauth.services.modules.mumble'`, to your `INSTALLED_APPS` list
- Append the following to your `local.py` settings file:

```
# Mumble Configuration
MUMBLE_URL = ""
```

### Overview

Mumble is a free voice chat server. While not as flashy as TeamSpeak, it has all the functionality and is easier to customize. And is better. I may be slightly biased.

### Dependencies

The mumble server package can be retrieved from a repository we need to add, mumble/release.

```
apt-add-repository ppa:mumble/release
apt-get update
```

Now two packages need to be installed:

```
apt-get install python-software-properties mumble-server
```

Download the appropriate authenticator release from [the authenticator repository](#) and install the python dependencies for it:

```
pip install -r requirements.txt
```

### Configuring Mumble

Mumble ships with a configuration file that needs customization. By default it's located at /etc/mumble-server.ini. Open it with your favourite text editor:

```
nano /etc/mumble-server.ini
```

**REQUIRED:** To enable the ICE authenticator, edit the following:

- `icesecretwrite=MY_CLEVER_PASSWORD`, obviously choosing a secure password
- ensure the line containing `Ice="tcp -h 127.0.0.1 -p 6502"` is uncommented

By default mumble operates on SQLite which is fine, but slower than a dedicated MySQL server. To customize the database, edit the following:

- uncomment the database line, and change it to `database=alliance_mumble`
- `dbDriver=QMYSQL`
- `dbUsername=allianceserver` or whatever you called the Alliance Auth MySQL user
- `dbPassword=` that user's password
- `dbPort=3306`
- `dbPrefix=murmur_`

To name your root channel, uncomment and set `registerName=` to whatever cool name you want

Save and close the file.

To get Mumble superuser account credentials, run the following:

```
dpkg-reconfigure mumble-server
```

Set the password to something you'll remember and write it down. This is needed to manage ACLs.

Now restart the server to see the changes reflected.

```
service mumble-server restart
```

That's it! Your server is ready to be connected to at `example.com:64738`

## Configuring the Authenticator

The ICE authenticator lives in the `mumble-authenticator` repository, `cd` to the directory where you cloned it.

Make a copy of the default config:

```
cp authenticator.ini.example authenticator.ini
```

Edit `authenticator.ini` and change these values:

- `[database]`
  - `user` = your allianceserver MySQL user
  - `password` = your allianceserver MySQL user's password
- `[ice]`
  - `secret` = the `icewritesecret` password set earlier

Test your configuration by starting it: `python authenticator.py`

## Running the Authenticator

The authenticator needs to be running 24/7 to validate users on Mumble. This can be achieved by adding a section to your auth project's supervisor config file like the following example:

```
[program:authenticator]
command=/path/to/venv/bin/python authenticator.py
directory=/path/to/authenticator/directory/
user=allianceserver
stdout_logfile=/path/to/authenticator/directory/authenticator.log
stderr_logfile=/path/to/authenticator/directory/authenticator.log
autostart=true
autorestart=true
startsecs=10
priority=998
```

Note that groups will only be created on Mumble automatically when a user joins who is in the group.

## Prepare Auth

In your project's settings file, set `MUMBLE_URL` to the public address of your mumble server. Do not include any leading `http://` or `mumble://`.

Run migrations and restart Gunicorn and Celery to complete setup.

### Openfire

Openfire is a Jabber (XMPP) server.

### Prepare Your Settings

- Add 'allianceauth.services.modules.openfire', to your INSTALLED\_APPS list
- Append the following to your auth project's settings file:

```
# Jabber Configuration
JABBER_URL = ""
JABBER_PORT = 5223
JABBER_SERVER = ""
OPENFIRE_ADDRESS = ""
OPENFIRE_SECRET_KEY = ""
BROADCAST_USER = ""
BROADCAST_USER_PASSWORD = ""
BROADCAST_SERVICE_NAME = "broadcast"
```

### Dependencies

Openfire require a Java 8 runtime environment.

Ubuntu:

```
apt-get install openjdk-8-jdk
```

CentOS:

```
yum -y install java-1.8.0-openjdk java-1.8.0-openjdk-devel
```

### Setup

#### Download Installer

Openfire is not available through repositories so we need to get a package from the developer.

On your PC, navigate to the [Ignite Realtime downloads](#) section, and under Openfire select Linux, click on the Ubuntu: Debian package (second from bottom of list, ends with .deb) or CentOS: RPM Package (no JRE bundled, as we have installed it on the host)

Retrieve the file location by copying the URL from the “click here” link, depending on your browser you may have a Copy Link or similar option in your right click menu.

In the console, ensure you're in your user's home directory: `cd ~`

Now download the package. Replace the link below with the link you got earlier.

```
wget https://www.igniterealtime.org/downloadServlet?filename=openfire/openfire_4.2.3_
↪all.deb
```

Now install from the package. Replace the filename with your filename (the last part of the download URL is the file name)

Ubuntu:

```
dpkg -i openfire_4.2.3_all.deb
```

CentOS:

```
yum install -y openfire-4.2.3-1.noarch.rpm
```

## Create Database

Performance is best when working from a SQL database. If you installed MySQL or MariaDB alongside your auth project, go ahead and create a database for Openfire:

```
mysql -u root -p
create database alliance_jabber;
grant all privileges on alliance_jabber . * to 'allianceserver'@'localhost';
exit;
```

## Web Configuration

The remainder of the setup occurs through Openfire's web interface. Navigate to `http://example.com:9090`, or if you're behind CloudFlare, go straight to your server's IP:9090.

Select your language. I sure hope it's English if you're reading this guide.

Under Server Settings, set the Domain to `example.com` replacing it with your actual domain. Don't touch the rest.

Under Database Settings, select `Standard Database Connection`

On the next page, select `MySQL` from the dropdown list and change the following:

- `[server]` is replaced by `127.0.0.1`
- `[database]` is replaced by the name of the database to be used by Openfire
- enter the login details for your auth project's database user

If Openfire returns with a failed to connect error, re-check these settings. Note the lack of square brackets.

Under Profile Settings, leave `Default` selected.

Create an administrator account. The actual name is irrelevant, just don't lose this login information.

Finally, log in to the console with your admin account.

Edit your auth project's settings file and enter the values you just set:

- `JABBER_URL` is the public address of your jabber server
- `JABBER_PORT` is the port for clients to connect to (usually 5223)
- `JABBER_SERVER` is the name of the jabber server. If you didn't alter it during install it'll usually be your domain (eg `example.com`)
- `OPENFIRE_ADDRESS` is the web address of Openfire's web interface. Use `http://` with port 9090 or `https://` with port 9091 if you configure SSL in Openfire

### REST API Setup

Navigate to the `plugins` tab, and then `Available Plugins` on the left navigation bar. You'll need to fetch the list of available plugins by clicking the link.

Once loaded, press the green plus on the right for `REST API`.

Navigate the `Server` tab, `Server Settings` subtab. At the bottom of the left navigation bar select `REST API`.

Select `Enabled`, and `Secret Key Auth`. Update your auth project's settings with this secret key as `OPENFIRE_SECRET_KEY`.

### Broadcast Plugin Setup

Navigate to the `Users/Groups` tab and select `Create New User` from the left navigation bar.

Pick a username (e.g. `broadcast`) and password for your ping user. Enter these in your auth project's settings file as `BROADCAST_USER` and `BROADCAST_USER_PASSWORD`. Note that `BROADCAST_USER` needs to be in the format `user@example.com` matching your jabber server name. Press `Create User` to save this user.

Broadcasting requires a plugin. Navigate to the `plugins` tab, press the green plus for the `Broadcast` plugin.

Navigate to the `Server` tab, `Server Manager` subtab, and select `System Properties`. Enter the following:

- Name: `plugin.broadcast.disableGroupPermissions`
  - Value: `True`
  - Do not encrypt this property value
- Name: `plugin.broadcast.allowedUsers`
  - Value: `broadcast@example.com`, replacing the domain name with yours
  - Do not encrypt this property value

If you have troubles getting broadcasts to work, you can try setting the optional (you will need to add it) `BROADCAST_IGNORE_INVALID_CERT` setting to `True`. This will allow invalid certificates to be used when connecting to the Openfire server to send a broadcast.

### Preparing Auth

Once all settings are entered, run migrations and restart Gunicorn and Celery.

### Group Chat

Channels are available which function like a chat room. Access can be controlled either by password or ACL (not unlike mumble).

Navigate to the `Group Chat` tab and select `Create New Room` from the left navigation bar.

- Room ID is a short, easy-to-type version of the room's name users will connect to
- Room Name is the full name for the room
- Description is short text describing the room's purpose
- Set a password if you want password authentication
- Every other setting is optional. Save changes.

Now select your new room. On the left navigation bar, select `Permissions`.

ACL is achieved by assigning groups to each of the three tiers: `Owners`, `Admins` and `Members`. `Outcast` is the blacklist. You'll usually only be assigning groups to the `Member` category.

## phpBB3

### Overview

phpBB is a free PHP-based forum.

### Dependencies

phpBB3 requires PHP installed in your web server. Apache has `mod_php`, NGINX requires `php-fpm`. See [the official guide](#) for PHP package requirements.

### Prepare Your Settings

In your auth project's settings file, do the following:

- Add `'allianceauth.services.modules.phpbb3'`, to your `INSTALLED_APPS` list
- Append the following to the bottom of the settings file:

```
# PHPBB3 Configuration
PHPBB3_URL = ''
DATABASES['phpbb3'] = {
    'ENGINE': 'django.db.backends.mysql',
    'NAME': 'alliance_forum',
    'USER': 'allianceserver-phpbb3',
    'PASSWORD': 'password',
    'HOST': '127.0.0.1',
    'PORT': '3306',
}
```

## Setup

### Prepare the Database

Create a database to install phpBB3 in.

```
mysql -u root -p
create database alliance_forum;
grant all privileges on alliance_forum . * to 'allianceserver'@'localhost';
exit;
```

Edit your auth project's settings file and fill out the `DATABASES['phpbb3']` part.

### Download phpBB3

phpBB3 is available as a zip from their website. Navigate to the website's [downloads section](#) using your PC browser and copy the URL for the latest version zip.

In the console, navigate to your user's home directory: `cd ~`

Now download using `wget`, replacing the URL with the URL for the package you just retrieved

```
wget https://www.phpbb.com/files/release/phpBB-3.2.2.zip
```

This needs to be unpackaged. Unzip it, replacing the file name with that of the file you just downloaded

```
unzip phpBB-3.2.2.zip
```

Now we need to move this to our web directory. Usually `/var/www/forums`.

```
mv phpBB3 /var/www/forums
```

The web server needs read/write permission to this folder

Apache: `chown -R www-data:www-data /var/www/forums` Nginx: `chown -R nginx:nginx /var/www/forums`

---

**Tip:** Nginx: Some distributions use the `www-data:www-data` user:group instead of `nginx:nginx`. If you run into problems with permissions try it instead.

---

### Configuring Web Server

You will need to configure you web server to serve PHPBB3 before proceeding with installation.

A minimal Apache config file might look like:

```
<VirtualHost *:80>
    ServerName forums.example.com
    DocumentRoot /var/www/forums
    <Directory /var/www/forums>
        Require all granted
        DirectoryIndex index.php
    </Directory>
</VirtualHost>
```

A minimal Nginx config file might look like:

```
server {
    listen 80;
    server_name forums.example.com;
    root /var/www/forums;
    index index.php;
    access_log /var/logs/forums.access.log;

    location ~ /(config\.php|common\.php|cache|files|images/avatars/
↪upload|includes|store) {
        deny all;
        return 403;
    }
}
```



```

location ~* \.(gif|jpe?g|png|css)$ {
    expires 30d;
}

location ~ /\.php$ {
    try_files $uri =404;
    fastcgi_pass unix:/tmp/php.socket;
    fastcgi_index index.php;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    include fastcgi_params;
}
}

```

Enter your forum's web address as the `PHPBB3_URL` setting in your auth project's settings file.

## Web Install

Navigate to your forums web address where you will be presented with an installer.

Click on the `Install` tab.

All the requirements should be met. Press `Start Install`.

Under Database Settings, set the following:

- Database Type is `MySQL`
- Database Server Hostname is `127.0.0.1`
- Database Server Port is left blank
- Database Name is `alliance_forum`
- Database Username is your auth MySQL user, usually `allianceserver`
- Database Password is this user's password

If you use a table prefix other than the standard `phpbb_` you need to add an additional setting to your auth project's settings file, `PHPBB3_TABLE_PREFIX = ''`, and enter the prefix.

You should see `Successful Connection` and proceed.

Enter administrator credentials on the next page.

Everything from here should be intuitive.

phpBB will then write its own config file.

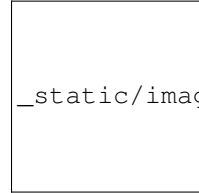
## Open the Forums

Before users can see the forums, we need to remove the install directory

```
rm -rf /var/www/forums/install
```

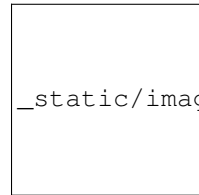
### Enabling Avatars

AllianceAuth sets user avatars to their character portrait when the account is created or password reset. We need to allow external URLs for avatars for them to behave properly. Navigate to the admin control panel for phpbb3, and under the General tab, along the left navigation bar beneath Board Configuration, select Avatar Settings. Set Enable Remote Avatars to Yes and then Submit.



\_static/images/installation/services/phpbb3/avatar\_settings.png

You can allow members to overwrite the portrait with a custom image if desired. Navigate to Users and Groups, Group Permissions, select the appropriate group (usually Member if you want everyone to have this ability), expand Advanced Permissions, under the Profile tab, set Can Change Avatars to Yes, and press Apply Permissions.



\_static/images/installation/services/phpbb3/avatar\_permissions.png

### Setting the default theme

Users generated via Alliance Auth do not have a default theme set. You will need to set this on the phpbb\_users table in SQL

```
mysql -u root -p
use alliance_forum;
alter table phpbb_users change user_style user_style int not null default 1
```

If you would like to use a theme that is NOT prosilver or theme “1”. You will need to deactivate prosilver, this will then fall over to the set forum wide default.

### Prepare Auth

Once settings have been configured, run migrations and restart Gunicorn and Celery.

### SMF

#### Overview

SMF is a free PHP-based forum.

## Dependencies

SMF requires PHP installed in your web server. Apache has `mod_php`, NGINX requires `php-fpm`. More details can be found in the [SMF requirements page](#).

## Prepare Your Settings

In your auth project's settings file, do the following:

- Add `'allianceauth.services.modules.smf'`, to your `INSTALLED_APPS` list
- Append the following to the bottom of the settings file:

```
# SMF Configuration
SMF_URL = ''
DATABASES['smf'] = {
    'ENGINE': 'django.db.backends.mysql',
    'NAME': 'alliance_smf',
    'USER': 'allianceserver-smf',
    'PASSWORD': 'password',
    'HOST': '127.0.0.1',
    'PORT': '3306',
}
```

## Setup

### Download SMF

Using your browser, you can download the latest version of SMF to your desktop computer. All SMF downloads can be found at [SMF Downloads](#). The latest recommended version will always be available at <http://www.simplerachines.org/download/index.php/latest/install/>. Retrieve the file location from the hyperlinked box icon for the zip full install, depending on your browser you may have a Copy Link or similar option in your right click menu.

Download using `wget`, replacing the URL with the URL for the package you just retrieved

```
wget https://download.simplerachines.org/index.php?thanks;filename=smf_2-0-15_install.
↪ zip
```

This needs to be unpackaged. Unzip it, replacing the file name with that of the file you just downloaded

```
unzip smf_2-0-15_install.zip
```

Now we need to move this to our web directory. Usually `/var/www/forums`.

```
mv smf /var/www/forums
```

The web server needs read/write permission to this folder

```
Apache:  chown -R www-data:www-data /var/www/forumsNginx:  chown -R nginx:nginx
/var/www/forums
```

**Tip:** Nginx: Some distributions use the `www-data:www-data` user:group instead of `nginx:nginx`. If you run into problems with permissions try it instead.

### Database Preparation

SMF needs a database. Create one:

```
mysql -u root -p
create database alliance_smf;
grant all privileges on alliance_smf . * to 'allianceserver'@'localhost';
exit;
```

Enter the database information into the DATABASES [ 'smf' ] section of your auth project's settings file.

### Web Server Configuration

Your web server needs to be configured to serve Alliance Market.

A minimal Apache config might look like:

```
<VirtualHost *:80>
    ServerName forums.example.com
    DocumentRoot /var/www/forums
    <Directory "/var/www/forums">
        DirectoryIndex index.php
    </Directory>
</VirtualHost>
```

A minimal Nginx config might look like:

```
server {
    listen 80;
    server_name forums.example.com;
    root /var/www/forums;
    index index.php;
    access_log /var/logs/forums.access.log;

    location ~ /\.php$ {
        try_files $uri =404;
        fastcgi_pass unix:/tmp/php.socket;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        include fastcgi_params;
    }
}
```

Enter the web address to your forums into the SMF\_URL setting in your auth project's settings file.

### Web Install

Navigate to your forums address where you will be presented with an installer.

Click on the Install tab.

All the requirements should be met. Press Start Install.

Under Database Settings, set the following:

- Database Type is MySQL

- Database Server Hostname is 127.0.0.1
- Database Server Port is left blank
- Database Name is `alliance_smf`
- Database Username is your auth MySQL user, usually `allianceserver`
- Database Password is this user's password

If you use a table prefix other than the standard `smf_` you need to add an additional setting to your auth project's settings file, `SMF_TABLE_PREFIX = ''`, and enter the prefix.

Follow the directions in the installer.

## Preparing Auth

Once settings are entered, apply migrations and restart Gunicorn and Celery.

## TeamSpeak 3

### Overview

TeamSpeak3 is the most popular VOIP program for gamers.

But have you considered using Mumble? Not only is it free, but it has features and performance far superior to Teamspeak3.

### Setup

Sticking with TS3? Alright, I tried.

### Prepare Your Settings

In your auth project's settings file, do the following:

- Add `'allianceauth.services.modules.teamspeak3'`, to your `INSTALLED_APPS` list
- Append the following to the bottom of the settings file:

```
# Teamspeak3 Configuration
TEAMSPEAK3_SERVER_IP = '127.0.0.1'
TEAMSPEAK3_SERVER_PORT = 10011
TEAMSPEAK3_SERVERQUERY_USER = 'serveradmin'
TEAMSPEAK3_SERVERQUERY_PASSWORD = ''
TEAMSPEAK3_VIRTUAL_SERVER = 1
TEAMSPEAK3_PUBLIC_URL = ''

CELERYBEAT_SCHEDULE['run_ts3_group_update'] = {
    'task': 'allianceauth.services.modules.teamspeak3.tasks.run_ts3_group_update',
    'schedule': crontab(minute='*/30'),
}
```

### Download Installer

To install we need a copy of the server. You can find the latest version from [this dl server](#) (I'd recommend getting the latest stable version – find this version number from the [TeamSpeak site](#)). Be sure to get a link to the Linux version.

Download the server, replacing the link with the link you got earlier.

```
http://dl.4players.de/ts/releases/3.1.1/teamspeak3-server_linux_amd64-3.1.1.tar.bz2
```

Now we need to extract the file.

```
tar -xf teamspeak3-server_linux_amd64-3.1.0.tar.bz2
```

### Create User

TeamSpeak needs its own user.

```
adduser --disabled-login teamspeak
```

### Install Binary

Now we move the server binary somewhere more accessible and change its ownership to the new user.

```
mv teamspeak3-server_linux_amd64 /usr/local/teamspeak  
chown -R teamspeak:teamspeak /usr/local/teamspeak
```

### Startup

Now we generate a startup script so TeamSpeak comes up with the server.

```
ln -s /usr/local/teamspeak/ts3server_startscript.sh /etc/init.d/teamspeak  
update-rc.d teamspeak defaults
```

Finally we start the server.

```
service teamspeak start
```

### Update Settings

The console will spit out a block of text. If it does not appear, it can be found with `service teamspeak status`. **SAVE THIS.**

If you plan on claiming the ServerAdmin token, do so with a different TeamSpeak client profile than the one used for your auth account, or you will lose your admin status.

Edit the settings you added to your auth project's settings file earlier, entering the following:

- `TEAMSPEAK3_SERVERQUERY_USER` is loginname from that block of text it just spat out (usually `serveradmin`)
- `TEAMSPEAK3_SERVERQUERY_PASSWORD` is password from that block of text it just spat out

- `TEAMSPEAK_VIRTUAL_SERVER` is the virtual server ID of the server to be managed - it will only ever not be 1 if your server is hosted by a professional company
- `TEAMSPEAK3_PUBLIC_URL` is the public address of your TeamSpeak server. Do not include any leading `http://` or `teamspeak://`

Once settings are entered, run migrations and restart Gunicorn and Celery.

## Generate User Account

And now we can generate ourselves a user account. Navigate to the services in Alliance Auth for your user account and press the checkmark for TeamSpeak 3.

Click the URL provided to automatically connect to our server. It will prompt you to redeem the serveradmin token, enter the `token` from startup.

## Groups

Now we need to make groups. AllianceAuth handles groups in teamspeak differently: instead of creating groups it creates an association between groups in TeamSpeak and groups in AllianceAuth. Go ahead and make the groups you want to associate with auth groups, keeping in mind multiple TeamSpeak groups can be associated with a single auth group.

Navigate back to the AllianceAuth admin interface (`example.com/admin`) and under `Services`, select `Auth / TS Groups`. In the top-right corner click `Add`.

The dropdown box provides all auth groups. Select one and assign TeamSpeak groups from the panels below. If these panels are empty, wait a minute for the database update to run, or see the [troubleshooting section](#) below.

## Troubleshooting

### Insufficient client permissions (failed on Invalid permission: 0x26)

Using the advanced permissions editor, ensure the Guest group has the permission `Use Privilege Keys` to gain permissions (under `Virtual Server` expand the `Administration` section)

To enable advanced permissions, on your client go to the `Tools` menu, `Application`, and under the `Misc` section, tick `Advanced permission system`

### TS group models not populating on admin site

The method which populates these runs every 30 minutes. To populate manually, start a django shell:

```
python manage.py shell
```

And execute the update:

```
from services.modules.teamspeak3.tasks import Teamspeak3Tasks
Teamspeak3Tasks.run_ts3_group_update()
```

Ensure that command does not return an error.

### 2564 access to default group is forbidden

This usually occurs because auth is trying to remove a user from the `Guest` group (group ID 8). The guest group is only assigned to a user when they have no other groups, unless you have changed the default teamspeak server config.

Teamspeak servers v3.0.13 and up are especially susceptible to this. Ensure the Channel Admin Group is not set to `Guest (8)`. Check by right clicking on the server name, `Edit virtual server`, and in the middle of the panel select the `Misc` tab.

### `TypeError: string indices must be integers, not str`

This error generally means teamspeak returned an error message that went unhandled. The full traceback is required for proper debugging, which the logs do not record. Please check the superuser notifications for this record and get in touch with a developer.

### 3331 flood ban

This most commonly happens when your teamspeak server is externally hosted. You need to add the auth server IP to the teamspeak serverquery whitelist. This varies by provider.

If you have SSH access to the server hosting it, you need to locate the teamspeak server folder and add the auth server IP on a new line in `server_query_whitelist.txt`

### 520 invalid loginname or password

The serverquery account login specified in `local.py` is incorrect. Please verify `TEAMSPEAK3_SERVERQUERY_USER` and `TEAMSPEAK3_SERVERQUERY_PASSWORD`. The [installation section](#) describes where to get them.

### 2568 insufficient client permissions

This usually occurs if you've created a separate serverquery user to use with auth. It has not been assigned sufficient permissions to complete all the tasks required of it. The full list of required permissions is not known, so assign liberally.

## XenForo

### Overview

[XenForo](#) is a popular paid forum. This guide will assume that you already have XenForo installed with a valid license (please keep in mind that XenForo is not free nor open-source, therefore you need to purchase a license first). If you come across any problems related with the installation of XenForo please contact their support service.

### Prepare Your Settings

In your auth project's settings file, do the following:

- Add `'allianceauth.services.modules.xenforo'`, to your `INSTALLED_APPS` list
- Append the following to your `local.py` settings file:



```
# XenForo Configuration
XENFORO_ENDPOINT = 'example.com/api.php'
XENFORO_DEFAULT_GROUP = 0
XENFORO_APIKEY = 'yourapikey'
```

## XenAPI

By default XenForo does not support any kind of API, however there is a third-party package called [XenAPI](#) which provides a simple REST interface by which we can access XenForo's functions in order to create and edit users.

The installation of XenAPI is pretty straight forward. The only thing you need to do is to download the `api.php` from the official repository and upload it in the root folder of your XenForo installation. The final result should look like this: `forumswebsite.com/api.php`

Now that XenAPI is installed the only thing left to do is to provide a key.

```
$restAPI = new RestAPI('REPLACE_THIS_WITH_AN_API_KEY');
```

## Configuration

The settings you created earlier now need to be filled out.

`XENFORO_ENDPOINT` is the address to the API you added. No leading `http://`, but be sure to include the `/api.php` at the end.

`XENFORO_DEFAULT_GROUP` is the ID of the group in XenForo auth users will be added to. Unfortunately XenAPI **cannot create new groups**, therefore you have to create a group manually and then get its ID.

`XENFORO_API_KEY` is the API key value you set earlier.

Once these are entered, run migrations and restart Gunicorn and Celery.

## 3.3 Maintenance

### 3.3.1 Your Auth Project

#### Overview

When installing Alliance Auth you are instructed to run the `allianceauth start` command which generates a folder containing your auth project. This auth project is based off Alliance Auth but can be customized how you wish.

#### The myauth Folder

The first folder created is the root directory of your auth project. This folder contains:

- the `manage.py` management script used to interact with Django
- a preconfigured `supervisor.conf` Supervisor config for running Celery (and optionally Gunicorn) automatically
- a `log` folder which contains log files generated by Alliance Auth

### The myauth Subfolder

Within your auth project root folder is another folder of the same name (a quirk of Django project structures). This folder contains:

- a Celery app definition in `celery.py` for registering tasks with the background workers
- a web server gateway interface script `wsgi.py` for processing web requests
- the root URL config `urls.py` which Django uses to direct requests to the appropriate view

There are also two subfolders for `static` and `templates` which allow adding new content and overriding default content shipped with Alliance Auth or Django.

And finally the settings folder.

### Settings Files

With the settings folder lives two settings files: `base.py` and `local.py`

The base settings file contains everything needed to run Alliance Auth. It handles configuration of Django and Celery, defines logging, and many other Django-required settings. This file should not be edited. While updating Alliance Auth you may be instructed to update the base settings file - this is achieved through the `allianceauth update` command which overwrites the existing base settings file.

The local settings file is referred to as “your auth project’s settings file” and you are instructed to edit it during the install process. You can add any additional settings required by other apps to this file. Upon creation the first line is `from .base import *` meaning all settings defined in the base settings file are loaded. You can override any base setting by simply redefining it in your local settings file.

### Log Files

Your auth project comes with four log file definitions by default. These are created in the `myauth/log/` folder at runtime.

- `allianceauth.log` contains all INFO level and above logging messages from Alliance Auth. This is useful for tracking who is making changes to the site, what is happening to users, and debugging any errors that may occur.
- `worker.log` contains logging messages from the Celery background task workers. This is useful for monitoring background processes such as group syncing to services.
- `beat.log` contains logging messages from the background task scheduler. This is of limited use unless the scheduler isn’t starting.
- `gunicorn.log` contains logging messages from Gunicorn workers. This contains all web-sourced messages found in `allianceauth.log` as well as runtime errors from the workers themselves.

When asking for assistance with your auth project be sure to first read the logs, and share any relevant entries.

### Custom Static and Templates

Within your auth project exists two folders named `static` and `templates`. These are used by Django for rendering web pages. Static refers to content Django does not need to parse before displaying, such as CSS styling or images. When running via a WSGI worker such as Gunicorn static files are copied to a location for the web server to read from. Templates are always read from the template folders, rendered with additional context from a view function, and then displayed to the user.

You can add extra static or templates by putting files in these folders. Note that changes to static requires running the `python manage.py collectstatic` command to copy to the web server directory.

It is possible to overload static and templates shipped with Django or Alliance Auth by including a file with the exact path of the one you wish to overload. For instance if you wish to add extra links to the menu bar by editing the template, you would make a copy of the `allianceauth/templates/allianceauth/base.html` file to `myauth/templates/allinceauth/base.html` and edit it there. Notice the paths are identical after the `templates/` directory - this is critical for it to be recognized. Your custom template would be used instead of the one included with Alliance Auth when Django renders the web page. Similar idea for static: put CSS or images at an identical path after the `static/` directory and they will be copied to the web server directory instead of the ones included.

## Custom URLs and Views

It is possible to add or override URLs with your auth project's URL config file. Upon install it is of the form:

```
import allianceauth.urls

urlpatterns = [
    url(r'', include(allianceauth.urls)),
]
```

This means every request gets passed to the Alliance Auth URL config to be interpreted.

If you wanted to add a URL pointing to a custom view, it can be added anywhere in the list if not already used by Alliance Auth:

```
import allianceauth.urls
import myauth.views

urlpatterns = [
    url(r'', include(allianceauth.urls)),
    url(r'myview/$', myauth.views.myview, name='myview'),
]
```

Additionally you can override URLs used by Alliance Auth here:

```
import allianceauth.urls
import myauth.views

urlpatterns = [
    url(r'account/login/$', myauth.views.login, name='auth_login_user'),
    url(r'', include(allianceauth.urls)),
]
```

## Adding and Removing Apps

Your auth project is just a regular Django project - you can add in [other Django apps](#) as desired. Most come with dedicated setup guides, but in general:

- add 'appname', to your `INSTALLED_APPS` setting
- run `python manage.py migrate`
- run `python manage.py collectstatic`

If you ever want to remove an app, you should first clear it from the database to avoid dangling foreign keys: `python manage.py migrate appname zero`. Then you can remove it from your auth project's `INSTALLED_APPS` list.

### 3.3.2 Troubleshooting

#### Something broken? Stuck on an issue? Can't get it set up?

Start by checking the [issues](#) - especially closed ones.

No answer?

- open an [issue](#)
- harass us on [gitter](#)

#### Logging

In its default configuration your auth project logs INFO and above messages to `myauth/log/allianceauth.log`. If you're encountering issues it's a good idea to view DEBUG messages as these greatly assist the troubleshooting process. These are printed to the console with manually starting the webserver via `python manage.py runserver`.

To record DEBUG messages in the log file, alter a setting in your auth project's settings file: `LOGGING['handlers']['log_file']['level'] = 'DEBUG'`. After restarting gunicorn and celery your log file will record all logging messages.

#### Common Problems

##### I'm getting an error 500 trying to connect to the website on a new install

*Great.* Error 500 is the generic message given by your web server when *anything* breaks. The actual error message is hidden in one of your auth project's log files. Read them to identify it.

##### Failed to configure log handler

Make sure the log directory is writeable by the `allianceserver` user: `chmown -R allianceserver:allianceserver /path/to/myauth/log/`, then restart the auth supervisor processes.

##### Groups aren't syncing to services

Make sure the background processes are running: `supervisorctl status myauth:.` If `myauth:worker` or `myauth:beat` do not show RUNNING read their log files to identify why.

##### Task queue is way too large

Stop celery workers with `supervisorctl stop myauth:worker` then clear the queue:

```
redis-cli FLUSHALL
celery -A myauth worker --purge
```

Press Control+C once.

Now start the worker again with `supervisorctl start myauth:worker`

### Proxy timeout when entering email address

This usually indicates an issue with your email settings. Ensure these are correct and your email server/service is properly configured.

### No images are available to users accessing the website

This is likely due to a permissions mismatch. Check the setup guide for your web server. Additionally ensure the user who owns `/var/www/myauth/static` is the same user as running your webserver, as this can be non-standard.

### Unable to execute ‘gunicorn myauth.wsgi’ or ImportError: No module named ‘myauth.wsgi’

Gunicorn needs to have context for its running location, `/home/allianceserver/myauth/gunicorn myauth.wsgi` will not work, instead `cd /home/allianceserver/myauth then gunicorn myauth.wsgi` is needed to boot Gunicorn. This is handled in the Supervisor config, but this may be encountered running Gunicorn manually for testing.

## 3.4 Development

### 3.4.1 Documentation

The documentation for Alliance Auth uses [Sphinx](#) to build documentation. When a new commit to specific branches is made (master, primarily), the repository is automatically pulled, docs built and deployed on [readthedocs.org](#).

Documentation was migrated from the GitHub wiki pages and into the repository to allow documentation changes to be included with pull requests. This means that documentation can be guaranteed to be updated when a pull request is accepted rather than hoping documentation is updated afterwards or relying on maintainers to do the work. It also allows for documentation to be maintained at different versions more easily.

### Building Documentation

If you’re developing new documentation, its likely you’ll want or need to test build it before committing to your branch. To achieve this you can use Sphinx to build the documentation locally as it appears on Read the Docs.

Activate your virtual environment (if you’re using one) and install the documentation requirements found in `docs/requirements.txt` using `pip`, e.g. `pip install -r docs/requirements.txt`.

You can then build the docs by changing to the `docs/` directory and running `make html` or `make dirhtml`, depending on how the Read the Docs project is configured. Either should work fine for testing. You can now find the output of the build in the `/docs/_build/` directory.

Occasionally you may need to fully rebuild the documents by running `make clean` first, usually when you add or rearrange toctrees.

### Documentation Format

CommonMark Markdown is the current preferred format, via [recommonmark](#). reStructuredText is supported if required, or you can execute snippets of reST inside Markdown by using a code block:

```
```eval_rst
reStructuredText here
```
```

Markdown is used elsewhere on Github so it provides the most portability of documentation from Issues and Pull Requests as well as providing an easier initial migration path from the Github wiki.

### 3.4.2 Integrating Services

One of the primary roles of Alliance Auth is integrating with external services in order to authenticate and manage users. This is achieved through the use of service modules.

#### The Service Module

Each service module is its own self contained Django app. It will likely contain views, models, migrations and templates. Anything that is valid in a Django app is valid in a service module.

Normally service modules live in `services.modules` though they may also be installed as external packages and installed via `pip` if you wish. A module is installed by including it in the `INSTALLED_APPS` setting.

#### Service Module Structure

Typically a service will contain 5 key components:

- *The Hook*
- *The Service Manager*
- *The Views*
- *The Tasks*
- *The Models*

The architecture looks something like this:

```

      urls ----- Views
          |           |
          |           |
ServiceHook ----- Tasks ----- Manager
          |
          |
AllianceAuth

Where:
  Module -- Dependency/Import
```

While this is the typical structure of the existing services modules, there is no enforcement of this structure and you are, effectively, free to create whatever architecture may be necessary. A service module need not even communicate with an external service, for example, if similar triggers such as `validate_user`, `delete_user` are required for a module it may be convenient to masquerade as a service. Ideally though, using the common structure improves the maintainability for other developers.

## The Hook

In order to integrate with Alliance Auth service modules must provide a `services_hook`. This hook will be a function that returns an instance of the `services.hooks.ServiceHook` class and decorated with the `@hooks.registerhook` decorator. For example:

```
@hooks.register('services_hook')
def register_service():
    return ExampleService()
```

This would register the `ExampleService` class which would need to be a subclass of `services.hooks.ServiceHook`.

---

**Important:** The hook **MUST** be registered in `yourservice.auth_hooks` along with any other hooks you are registering for Alliance Auth.

---

A subclassed `ServiceHook` might look like this:

```
class ExampleService(ServicesHook):
    def __init__(self):
        ServicesHook.__init__(self)
        self.urlpatterns = urlpatterns
        self.service_url = 'http://exampleservice.example.com'

    """
    Overload base methods here to implement functionality
    """
```

## The ServiceHook class

The base `ServiceHook` class defines function signatures that Alliance Auth will call under certain conditions in order to trigger some action in the service.

You will need to subclass `services.hooks.ServiceHook` in order to provide implementation of the functions so that Alliance Auth can interact with the service correctly. All of the functions are optional, so its up to you to define what you need.

Instance Variables:

- `self.name`
- `self.urlpatterns`
- `self.service_ctrl_template`

Properties:

- `title`

Functions:

- *delete\_user*
- *validate\_user*
- *sync\_nickname*
- *update\_groups*
- *update\_all\_groups*
- *service\_enabled\_members*
- *service\_enabled\_blues*
- *service\_active\_for\_user*
- *show\_service\_ctrl*
- *render\_service\_ctrl*

### **self.name**

Internal name of the module, should be unique amongst modules.

### **self.urlpatterns**

You should define all of your service URLs internally, usually in `urls.py`. Then you can import them and set `self.urlpatterns` to your defined urlpatterns.

```
from . import urls
...
class MyService(ServiceHook):
    def __init__(self):
        ...
        self.urlpatterns = urls.urlpatterns
```

All of your apps defined urlpatterns will then be included in the `URLconf` when the core application starts.

### **self.service\_ctrl\_template**

This is provided as a courtesy and defines the default template to be used with *render\_service\_ctrl*. You are free to redefine or not use this variable at all.

### **title**

This is a property which provides a user friendly display of your service's name. It will usually do a reasonably good job unless your service name has punctuation or odd capitalisation. If this is the case you should override this method and return a string.

### **delete\_user**

```
def delete_user(self, user, notify_user=False):
```



Delete the users service account, optionally notify them that the service has been disabled. The `user` parameter should be a Django User object. If `notify_user` is set to `True` a message should be set to the user via the `notifications` module to alert them that their service account has been disabled.

The function should return a boolean, `True` if successfully disabled, `False` otherwise.

### `validate_user`

```
def validate_user(self, user):
```

Validate the users service account, deleting it if they should no longer have access. The `user` parameter should be a Django User object.

An implementation will probably look like the following:

```
def validate_user(self, user):
    logger.debug('Validating user %s %s account' % (user, self.name))
    if ExampleTasks.has_account(user) and not self.service_active_for_user(user):
        self.delete_user(user, notify_user=True)
```

No return value is expected.

This function will be called periodically on all users to validate that the given user should have their current service accounts.

### `sync_nickname`

```
def sync_nickname(self, user):
```

Very optional. As of writing only one service defines this. The `user` parameter should be a Django User object. When called, the given users nickname for the service should be updated and synchronised with the service.

If this function is defined, an admin action will be registered on the Django Users view, allowing admins to manually trigger this action for one or many users. The hook will trigger this action user by user, so you won't have to manage a list of users.

### `update_groups`

```
def update_groups(self, user):
```

Update the users group membership. The `user` parameter should be a Django User object. When this is called the service should determine the groups the user is a member of and synchronise the group membership with the external service. If you service does not support groups then you are not required to define this.

If this function is defined, an admin action will be registered on the Django Users view, allowing admins to manually trigger this action for one or many users. The hook will trigger this action user by user, so you won't have to manage a list of users.

This action is usually called via a signal when a users group membership changes (joins or leaves a group).

### `update_all_groups`

```
def update_all_groups(self):
```

The service should iterate through all of its recorded users and update their groups.

I'm really not sure when this is called, it may have been a hold over from before signals started to be used. Regardless, it can be useful to server admins who may call this from a Django shell to force a synchronisation of all user groups for a specific service.

### service\_active\_for\_user

```
def service_active_for_user(self, user):
```

Is this service active for the given user? The `user` parameter should be a Django User object.

Usually you wont need to override this as it calls `service_enabled_members` or `service_enabled_blues` depending on the users state.

### show\_service\_ctrl

```
def show_service_ctrl(self, user, state):
```

Should the service be shown for the given user with the given state? The `user` parameter should be a Django User object, and the `state` parameter should be a valid state from `authentication.states`.

Usually you wont need to override this function.

For more information see the [render\\_service\\_ctrl](#) section.

### render\_service\_ctrl

```
def render_services_ctrl(self, request):
```

Render the services control row. This will be called for all active services when a user visits the `/services/` page and `show_service_ctrl` returns True for the given user.

It should return a string (usually from `render_to_string`) of a table row (`<tr>`) with 4 columns (`<td>`). Column #1 is the service name, column #2 is the users username for this service, column #3 is the services URL, and column #4 is the action buttons.

You may either define your own service template or use the default one provided. The default can be used like this example:

```
def render_services_ctrl(self, request):
    """
    Example for rendering the service control panel row
    You can override the default template and create a
    custom one if you wish.
    :param request:
    :return:
    """
    urls = self.Urns()
    urls.auth_activate = 'auth_example_activate'
    urls.auth_deactivate = 'auth_example_deactivate'
    urls.auth_reset_password = 'auth_example_reset_password'
    urls.auth_set_password = 'auth_example_set_password'
    return render_to_string(self.service_ctrl_template, {
        'service_name': self.title,
        'urls': urls,
        'service_url': self.service_url,
        'username': 'example username'
    }, request=request)
```

---

the `Urls` class defines the available URL names for the 4 actions available in the default template:

- Activate (create service account)
- Deactivate (delete service account)
- Reset Password (random password)
- Set Password (custom password)

If you don't define one or all of these variable the button for the undefined URLs will not be displayed.

Most services will survive with the default template. If, however, you require extra buttons for whatever reason, you are free to provide your own template as long as you stick within the 4 columns. Multiple rows should be OK, though may be confusing to users.

### Menu Item Hook

If your services needs cannot be satisfied by the Service Control row, you are free to specify extra hooks by subclassing or instantiating the `services.hooks.MenuItemHook` class.

For more information see the [Menu Hooks](#) page.

### The Service Manager

The service manager is what interacts with the external service. Ideally it should be completely agnostic about its environment, meaning that it should avoid calls to Alliance Auth and Django in general (except in special circumstances where the service is managed locally, e.g. Mumble). Data should come in already arranged by the Tasks and data passed back for the tasks to manage or distribute.

The reason for maintaining this separation is that managers may be reused from other sources and there may not even be a need to write a custom manager. Likewise, by maintaining this neutral environment others may reuse the managers that we write. It can also significantly ease the unit testing of services.

### The Views

As mentioned at the start of this page, service modules are fully fledged Django apps. This means you're free to do whatever you wish with your views.

Typically most traditional username/password services define four views.

- Create Account
- Delete Account
- Reset Password
- Set Password

These views should interact with the service via the Tasks, though in some instances may bypass the Tasks and access the manager directly where necessary, for example OAuth functionality.

### The Tasks

The tasks component is the glue that holds all of the other components of the service module together. It provides the function implementation to handle things like adding and deleting users, updating groups, validating the existence of a users account. Whatever tasks `auth_hooks` and `views` have with interacting with the service will probably live here.

### The Models

Its very likely that you'll need to store data about a users remote service account locally. As service modules are fully fledged Django apps you are free to create as many models as necessary for persistent storage. You can create foreign keys to other models in Alliance Auth if necessary, though I *strongly* recommend you limit this to the User and Groups models from `django.contrib.auth.models` and query any other data manually.

If you create models you should create the migrations that go along with these inside of your module/app.

### Examples

There is a bare bones example service included in `services.modules.example`, you may like to use this as the base for your new service.

You should have a look through some of the other service modules before you get started to get an idea of the general structure of one. A lot of them aren't perfect so don't feel like you have to rigidly follow the structure of the existing services if you think its sub-optimal or doesn't suit the external service you're integrating.

### Testing

You will need to add unit tests for all aspects of your service module before it is accepted. Be mindful that you don't actually want to make external calls to the service so you should mock the appropriate components to prevent this behaviour.

## 3.4.3 Menu Hooks

The menu hooks allow you to dynamically specify menu items from your plugin app or service. To achieve this you should subclass or instantiate the `services.hooks.MenuItemHook` class and then register the menu item with one of the hooks.

To register a `MenuItemHook` class you would do the following:

```
@hooks.register('menu_item_hook')
def register_menu():
    return MenuItemHook('Example Item', 'glyphicon glyphicon-heart', 'example_url_name
↪', 150)
```

The `MenuItemHook` class specifies some parameters/instance variables required for menu item display.

`MenuItemHook(text, classes, url_name, order=None)`

#### text

The text value of the link

### classes

The classes that should be applied to the bootstrap menu item icon

### url\_name

The name of the Django URL to use

### order

An integer which specifies the order of the menu item, lowest to highest

### navactive

A list of views or namespaces the link should be highlighted on. See [django-navhelper](#) for usage. Defaults to the supplied `url_name`.

If you cannot get the menu item to look the way you wish, you are free to subclass and override the default render function and the template used.

## 3.4.4 URL Hooks

---

**Note:** URLs added through URL Hooks are protected by a decorator which ensures the requesting user is logged in and has a main character set.

---

The URL hooks allow you to dynamically specify URL patterns from your plugin app or service. To achieve this you should subclass or instantiate the `services.hooks.UrlHook` class and then register the URL patterns with the hook.

To register a `UrlHook` class you would do the following:

```
@hooks.register('url_hook')
def register_urls():
    return UrlHook(app_name.urls, 'app_name', r'^app_name/')
```

The `UrlHook` class specifies some parameters/instance variables required for URL pattern inclusion.

`UrlHook(urls, app_name, base_url)`

### urls

The urls module to include. See [the Django docs](#) for designing urlpatterns.

### namespace

The URL namespace to apply. This is usually just the app name.

### base\_url

The URL prefix to match against in regex form. Example `r'^app_name/'`. This prefix will be applied in front of all URL patterns included. It is possible to use the same prefix as existing apps (or no prefix at all) but [standard URL resolution](#) ordering applies (hook URLs are the last ones registered).

### Example

An app called `plugin` provides a single view:

```
def index(request):  
    return render(request, 'plugin/index.html')
```

The app's `urls.py` would look like so:

```
from django.conf.urls import url  
import plugin.views  
  
urlpatterns = [  
    url(r'^index$', plugin.views.index, name='index'),  
]
```

Subsequently it would implement the `UrlHook` in a dedicated `auth_hooks.py` file like so:

```
from alliance_auth import hooks  
from services.hooks import UrlHook  
import plugin.urls  
  
@hooks.register('url_hook')  
def register_urls():  
    return UrlHook(plugin.urls, 'plugin', r'^plugin/')
```

When this app is included in the project's settings, `INSTALLED_APPS` users would access the index view by navigating to `https://example.com/plugin/index`.