# Anitya Documentation

*Release 1.0.0*

**Red Hat, Inc. and others**

**Jan 20, 2021**

# Contents

Anitya is a release monitoring project.

Its goal is to regularly check if a project has made a new release. When Anitya discovers a new release for a project, it publishes a RabbitMQ message via fedora messaging. This makes it easy to integrate with Anitya and perform actions when a new release is created for a project. For example, the Fedora project runs a service called the-new-hotness which files a Bugzilla bug against a package when the upstream project makes a new release.

Anitya provides a user-friendly interface to add, edit, or browse projects.

**Github page** https://github.com/fedora-infra/anitya

CHAPTER 1

# User Guide

## 1.1 User Guide

Anitya attempts to select reasonable default settings for projects and in many cases, these should work just fine. However, there are millions of software projects out there and some do not follow common release methods. In those cases, Anitya offers more flexible tools.

In this chapter of documentation you can find use cases that could be performed by user in Anitya.

For use cases which needs admin permissions, please refer to *Admin User Guide*.

### 1.1.1 Logging in

To log in to Anitya you need to have an account. Anitya itself doesn't provide any way to create an account, but it provides option to log in with 3rd party account. Currently you can log in with either Fedora account or use custom openid server.

### 1.1.2 Logging out

If you are currently logged in Anitya, you can log out by clicking on logout link in the page header.

### 1.1.3 Listing Projects

Anitya provides multiple ways of listing the available projects. All options are accessible from *projects* menu in site header. You don't need to be logged in to list the projects.

From this menu you can choose from these options:

- all will show you table containing all the projects available in Anitya sorted by name.
- updated will show you table containing all the projects that Anitya checked without issue. This table is sorted by the time the last check was done.

- failed to update will show you table containing all the projects that Anitya failed to check for new version. On this page you can find number of unsuccessful attempts and the reason for the error. This table is sorted by number of unsuccessful attempts.
- never updated will show you table containing all the projects that Anitya never checked without an issue. In most cases this means that the project is incorrectly set up. They are ordered by the date of creation with the oldest on the top of table.
- archived will show you projects, that are archived and Anitya no longer checks for new versions of these projects. The archived project can no longer be edited, but it is good to have them available for history.

Each option allows you to filter the list by using fields above the table. These fields are supporting patterns. For example you can search for projects ending with `py` by typing `*py`.

### 1.1.4 Finding Project by Name

To find project by name quickly you can use a search field in top of the page. This field supports pattern recognition. For example you can search for projects ending with `py` by typing `*py`.

### 1.1.5 Creating New Project

You can create new projects in the web interface if you are logged in. When you create a new project, you must select a *backend* to fetch the version with. If the project is released in several places, please use the backend for the language *ecosystem*. For example, if a project is hosted on GitHub and publishes releases to PyPI, use the PyPI backend.

---

**Note:** Occasionally projects stop publishing releases to their ecosystem's package index, but continue to tag releases. In those cases, it's fine to use the backend for the source hosting service (GitHub, BitBucket, etc.) the project is using.

---

Bellow you will find the fields currently available on project with description.

#### Project Name

The project name should match the upstream project name. Duplicate project names are allowed as long as the projects are not part of the same ecosystem. That is, you can have two projects called `msgpack`, but you cannot have two projects called `msgpack` that are both in the `PyPI` ecosystem.

---

**Note:** When project is not part of any ecosystem, duplicate projects are detected based on the homepage of project.

---

#### Homepage

Homepage of the project. If the project doesn't have any homepage you could use the URL where project is hosted.

#### Backends

The backend of a project tells Anitya how to retrieve the versions of the project.

The backends available are:

- **BitBucket** for projects hosted on BitBucket

  You need to provide **BitBucket owner/project**. For example if project is hosted on *https://bitbucket.org/zzzeek/sqlalchemy* this field needs to contain *zzzeek/sqlalchemy*

- **CPAN** for perl projects hosted on CPAN
- **CRAN** for R projects hosted on CRAN
- **crates.io** for crates hosted on crates.io
- **Debian project** for projects hosted on the Debian ftp
- **Drupal6** for Drupal6 modules hosted on drupal.org
- **Drupal7** for Drupal7 modules hosted on drupal.org
- **folder** for projects whose release archives are provided in a basic apache folder or modified one.

  You need to provide **Version URL** where the archives could be found.

- **Freshmeat** for projects hosted on freshmeat.net / freecode.com
- **GitHub** for projects hosted on github.com. This backend is using Github v4 API.

  You need to provide **GitHub owner/project**. For example if project is hosted on *https://github.com/zzzeek/sqlalchemy* this field needs to contain *zzzeek/sqlalchemy*.

  When the **GitHub** backend is selected you can also check the option to **Check releases instead of tags**, this will tell Anitya to retrieve GitHub releases instead of tags. It could be helpful when project is using tags for other things than just releases.

- **GitLab** for projects hosted on GitLab server. This backend is using GitLab API v4.

  You need to provide **GitLab project url** which needs to point to project root on GitLab server.

- **GNOME** for projects hosted on download.gnome.org
- **GNU Project** for projects hosted on gnu.org
- **Google code** for projects hosted on code.google.com
- **Hackage** for projects hosted on hackage.haskell.org
- **Launchpad** for projects hosted on launchpad.net
- **Maven Central** for projects hosted on maven.org
- **npmjs** for projects hosted on npmjs.org
- **Packagist** for projects hosted on packagist.org

  You need to provide **Packagist owner/group**. For example if project is hosted on *https://packagist.org/packages/phpunit/php-code-coverage* this field needs to contain *phpunit/php-code-coverage*

- **pagure** for projects hosted on pagure.io
- **PEAR** for projects hosted on pear.php.net
- **PECL** for projects hosted on pecl.php.net
- **PyPI** for projects hosted on pypi.python.org
- **Rubygems** for projects hosted on rubygems.org

- **Sourcefoge** for projects hosted on sourceforge.net

  You need to provide **Sourceforge name** if the name on RSS feed is different then the project name on Sourceforge.

- **Stackage** for projects hosted on www.stackage.org

If your project cannot be used with any of these backend you can always try the **custom** backend. **custom** backend is for projects who require a more flexible way of finding their version.

The custom backend requires two arguments:

- **Version URL** is the url of the page where the versions information can be found, for example for banshee that would be their download page

---

**Note:** It's possible to provide a "glob" for projects that place their tarballs in multiple directories. For example, gcc uses `https://ftp.gnu.org/gnu/gcc/*/` to find the tarballs in each version directory.

---

- **Regex** is a regular expression using the Python re syntax to find the releases on the **Version URL** page.

---

**Note:** In most cases, you can set the **Regex** to *DEFAULT* which will make Anitya use its default regular expression:

```
(?i){project name}(?:[-_]?(?:minsrc|src|source))?[-_]([^-/_\s]+?)(?:[-_](?
↪:minsrc|src|source|asc|release))?\.(?:tar|t[bglx]z|tbz2|zip)
```

---

### Version Scheme

Version scheme is used for sorting the retrieved versions for the projects. Anitya provides three different versions scheme.

- **RPM** corresponds to versioning used by RPM packages
- **Calendar** corresponds to project versions in format described on Calendar Versioning
- **Semantic** corresponds to project versions in format described on Semantic Versioning 2.0.0

---

**Note:** Anitya currently doesn't work well with projects that are using multiple versions schemes throughout their life. For these projects just use the most recent scheme and the rest will be moved to bottom unsorted.

---

### Version Prefix

The version prefix can be used to retrieve the exact version number when the upstream maintainer prefixes its versions.

For example, if the project's version are: `foo-1.2`, you can set the version prefix to `foo-` to tell Anitya how to get the version `1.2`.

You can specify multiple prefixes by separating them by `;`. For example `foo-;v` will remove both `foo-` and `v` from retrieved versions.

---

**Note:** It's common for projects to prefix their source control tags with a `v` when making a release. Anitya will automatically strip this from versions it finds.

---

More concrete examples:

- junit tags are of the form: `r<version>`, to retrieve the version, one can set the version prefix to `r`.
- fdupes tags are of the form `fdupes-<version>`, for this project, the version prefix can be set to `fdupes-`.

### Pre-release filter

Sometimes the recognition of stable and unstable versions by *Version Scheme* isn't working for the project and in this field you can specify strings that will be considered as unstable release.

For example, if the project's version are: `1.2alpha`, you can set the pre-release filter to `alpha` to tell Anitya to treat this release as unstable.

You can specify multiple filters by separating them by `;`. For example `alpha;beta` will mark versions with `alpha` and `beta` as unstable.

---

**Note:** This filter is applied after recognition of unstable versions by *Version Scheme*. So the filter is not needed for cases where *Version Scheme* is able to recognize unstable versions.

---

### Version filter

Sometimes the upstream project is tagging things that aren't releases. For this case you can use this field to specify which version shouldn't be retrieved by Anitya.

For example, if the project has tag `xyz`, you can set the version filter to `xyz` to tell Anitya to ignore this tag.

You can specify multiple filters by separating them by `;`. For example `notrelease;test` will ignore versions with `notrelease` and `test`.

---

**Note:** This filter is not applied on version that is already retrieved, but if you create the filter and the admin deletes the version, it will not be retrieved again.

---

### Regular Expressions

Sometimes you need to use a custom regular expression to find the version on a page. Anitya accepts user-defined regular expressions using the Python re syntax. This option is only available when using **custom** backend.

The simplest way to check your regular expression is to open a python shell and test it.

Below is an example on how it can be done:

```
>>> url = 'http://www.opendx.org/download.html'
>>>
>>> import requests
>>> import re
>>> text = requests.get(url).text
>>> re.findall('version.is ([\d]*)\.', text)
[u'4']
>>> re.findall('version.is ([\d\.-]*)\.', text)
[u'4.4.4']
```

If you prefer graphical representation you can use Debuggex.

The regular expression `version.is ([\d\.]*)\.` can then be provided to anitya and used to find the new releases.

---

**Note:** Only the captured groups are used as version, delimited by dot. For example: `1_2_3` could be captured by regular expression `(\d)_(\d)_(\d)`. This will create version `1.2.3`.

---

#### Check latest release on submit

This option will tell the Anitya to do a check for versions when the project is submitted. In other case the project is scheduled and checked on next run check service, this service is doing checks regularly multiple times a day.

#### Distro (optional)

When creating a new project in Anitya you can specify distribution in which the project could be found. You can select from distributions that are already available in Anitya or you can create a new one later, see *Creating a New Distribution*.

#### Package (optional)

If you selected the distribution you must write the package name under which the project is known in distribution. For example python projects in Fedora distribution are prefixed with `python3-` prefix, so the project *alembic* will be called `python3-alembic` in Fedora. However the name of the project in Anitya should still be *alembic*.

#### Test check

You can test your changes before submitting by using the **Test check** button at the bottom of the project form. This will take the current values from the fields and do a check for new version with temporary project. This is very useful if you are doing multiple changes and you are not sure of the output.

### 1.1.6 Editing Project

You can edit any project in the web interface if you have logged in. The editing of a project could be done from the project page by clicking on button **Edit**.

The editing is very similar to creating a new project with the exception of optional fields which are missing. For field reference please check *Creating New Project*.

### 1.1.7 Creating New Distribution Mapping

To add a new mapping of the project to distribution you can use **Add new distribution mapping** under the *Mappings* table on project page.

This opens a new page which allows you to add a new mapping for the distribution of your choice.

#### Distribution

You can select from distributions that are already available in Anitya or you can create a new one and add the mapping later, see *Creating a New Distribution* for how to create a new distribution.

#### Package name

If you selected the distribution you must write the package name under which the project is known in distribution. For example python projects in Fedora distribution are prefixed with `python3-` prefix, so the project *alembic* will be called `python3-alembic` in Fedora.

### 1.1.8 Editing Distribution Mapping

To edit existing mapping of the project to distribution you can use **Edit** button beside corresponding mapping in the *Mappings* table on project page.

This opens a new page which is similar to *Creating New Distribution Mapping* page. See *Creating New Distribution Mapping* for more info about fields.

### 1.1.9 Flagging a Project

If you find a project which contains bad version, is duplicate, or is no longer supported upstream, you can flag this project. To flag a project you can use **Flag** button in top of the project page. This will redirect you to *Flag project* form, where you need to provide a reason. The flagged project will be later reviewed by admin user.

### 1.1.10 Listing Distributions

Anitya provides a way to look at all the distributions that Anitya knows about and that could be used when working with project mapping. To list all the distributions just click on the distros link in the header of page. This will show you a page with list of all the distributions sorted by name.

### 1.1.11 Creating a New Distribution

If you are missing any distribution in Anitya you can add it. To add a new distribution first list the existing distributions, see *Listing Distributions*, and then click Add a new distribution button. This will redirect you to a new page where you can fill out a distribution name and submit the new distribution.

### 1.1.12 Obtaining an API Token

If you need to communicate through API with Anitya (see *API Documentation* for more info) you will need an API token for any operation that is changing data in Anitya. To obtain one, you need to click on the settings link in page header. This will redirect you to your user settings page. Here you can see your User Id, which could be needed by admin user for some use cases, and API tokens. You can create a new token here, just provide some description (optional) and click **Create API Token** button.

### 1.1.13 Removing an API Token

If you wish to remove an existing API token created for your account, you need to click on the settings link in page header. This will redirect you to your user settings page. Here you can see your User Id, which could be needed by admin user for some use cases, and API tokens. You can remove a existing token here, just click **Remove API Token** button beside the API token you want to remove.

### 1.1.14 Reporting Issues

You can report problems on our issue tracker. The source code is also available on GitHub. The development team hangs out in `#fedora-apps` on the freenode network. Please do stop by and say hello.

## 1.2 Admin User Guide

This guide is intended for people, who have been granted administrator rights in Anitya. It will describe the use cases that can be done by the administrator user.

### 1.2.1 Delete Project

As admin in Anitya you have rights to delete the project, but this should be done only if you are sure that this project has no value. To delete a project, go on the project page and click **Delete** button in bottom of the project table.

### 1.2.2 Test Check on Project

As admin you can do a test check for releases on the project. This is done by clicking **Test check** button in top of the project table beside latest version. This will do a check for new releases and shows you the output of the operation in pop-up window. This operation will not change anything in the database, just shows you what will happen when next check for this project will be done.

### 1.2.3 Full Check on Project

As admin you can do a manual check for releases on the project. This is done by clicking **Full check** button in top of the project table beside latest version. This will do a check for new releases and saves the result. It's recommended to always do a *Test Check on Project* before.

### 1.2.4 Archive Project

As admin you can archive project. The archived project could no longer be edited and is not checked for a new releases, but will remain read-only in Anitya database. The archivation could be done by clicking the **Archive** button in bottom of project table on project page. This option is particularly useful for projects that are no longer alive upstream.

### 1.2.5 Unarchive Project

As admin you can unarchive archived projects. This could be done by clicking the **Unarchive** button in bottom of archived project table on project page.

### 1.2.6 Delete a Version on Project

As admin you have a power to delete any version on project. This could be useful if the project was incorrectly setup and retrieved something that is not considered a release. To delete a version click on the **[x]** beside the desired version in versions table on project page.

### 1.2.7 Delete All Versions on Project

It's sometimes easier to just delete all the versions and do a *Full Check on Project* again. In this case the admin could delete all versions at once by clicking **Delete versions** button in bottom of the versions table on project page. Be aware that this could send plenty of messages, because the message is sent every time a version is deleted.

### 1.2.8 Delete Mapping on Project

As admin you can delete a mapping on the project. This could be needed if the project is renamed in the distribution or no longer provided by the distribution. To delete the mapping click on the **Delete** button beside the mapping you want to delete in the mappings table on the project info page.

### 1.2.9 Delete Distribution

As admin you can delete a distribution in Anitya. This could be useful if there is distribution containing typo and the correct distribution is already in Anitya. It is always good to check if the distribution doesn't have any project mapped to it, to check click on the distros link in header and then click on the distribution you plan to delete. If there is no project mapped to this distribution it could be safely deleted. Otherwise the project mappings will be deleted with the distribution.

To delete the distribution click on the distros link in page header and then click on **Delete** button on the row which contains distribution you want to delete.

### 1.2.10 Edit an Existing Distribution

As admin you can edit a name of existing distribution in Anitya. To edit the distribution name clink on the distros link in header and then click on **Edit** button bellow the name of distribution you want to edit.

---

**Note:** The name must be unique, so if you change the name to distribution that already exists in Anitya, it will not be allowed.

---

### 1.2.11 User management

As admin user you are allowed to do a basic user management. This includes banning a user, giving admin rights or revoke them. To get to user management page click users link in page header. You can find some basic info about the users like *User Id*, *Username*, *E-mail*, if the user is currently admin and if the user is active or not.

#### Ban user

Sometimes there is a situation that needs the user to be banned from Anitya. If this is ever needed you can do it by clicking the **Ban** button in the row with desired user. This user is then marked as inactive and could no longer login to Anitya. The user needs to have *Active* set to *True* otherwise this action is not available.

**Note:** This option should be used as last resort.

#### Remove ban

To remove ban from a user click on the **Remove ban** button in the row with desired user. The user will be marked as active and will be able to login to Anitya again. The user needs to have *Active* set to *False* otherwise this action is not available.

#### Give admin rights

To give admin rights to another user click **Give admin permissions** button in the row with desired user. The user needs to have *Admin* set to *False* otherwise this action is not available.

#### Revoke admin rights

To revoke admin rights from another user click **Revoke admin permissions** button in the row with desired user. The user needs to have *Admin* set to *True* otherwise this action is not available.

**Note:** Admin user which is specified in Anitya configuration file can't be striped of admin rights this way.

### 1.2.12 Solving Flags

The flags could be solved on Flags page, which is accessed through flags link in the page header.

#### Basics

- try to not remove anything that could be of value
- when removing something, try to create as little impact for others as possible
- check flags once in a while
- if in doubt, ask in #fedora-apps on freenode IRC

#### Cases

Below are a number of common flags submitted by users. Please don't view these as rigid laws, but as guidelines to make our handling of cases consistent. Feel free to adapt them, if it seems reasonable.

#### Project X is a duplicate!

1. check if this actually is a duplicate; if not, close the flag
2. check which name matches the name used by the project itself
3. check which one seems more correct/complete (i.e. version source, distro mappings, . . . )
4. transfer valuable information/settings from to-be-deleted project to will-stay project, if any

5. delete the project duplicate, close the flag

**Note:** The flag will be deleted automatically if the flagged project is the one that will be deleted.

### pypi/gem/... duplicates

Two projects: "nispor" on GitHub, "nispor" on pypi/gem/whatever.

1. check if there is any value added by the pypi/gem/... package (e.g. the versions are different, pypi lags behind, tagging on github are unreliable ...). If there is, do nothing.
2. transfer valuable information/settings from github project to pypi project, if any
3. if there is no added value, delete the pypi one

### "Please delete this thing, was just testing"

1. check if this seems reasonable
2. delete the thing (project, mapping, version, ...).

### Package does not exist in distribution, please delete mapping

Check if it really doesn't exist, then:

a) does exist: do nothing

b) doesn't exist: delete the mapping

c) existed in old, non-EOL version: do nothing

d) existed in old, EOL version: delete

### User's flag is wrong, user misunderstood, ...

Just close. If you feel very motivated or it seems important, mail them and try to clear it up.

### Upstream dead

1. Check if the project is really dead upstream

   Last commit date is usually a good thing to check.

2. Archive the project

   This will lower the amount of projects which are checked for new versions, but the project will be still in Anitya, if anybody would want to look at it.

## 1.3 Release Notes

### 1.3.1 1.0.0 (2021-01-20)

#### API Changes

- Add versions resource to API v2 (#491)
- API v1 api/version/get is now returning only versions field instead of whole project when no version is retrieved (#898)

#### Features

- Add missing methods to fedora messaging schema (version 1.1.0) (PR#906)
- Add preview mode (#491)
- Allow removing all versions at once (admin only) (#623)
- Implement fedmsg meta methods in fedora messaging schema (#752)
- Flag pre-release versions (#753)
- Anitya should report every found version, not only latest (#774)
- Add option to archive and unarchive project (admin only) (#865)
- Add version filter to project (#898)

#### Bug Fixes

- Yahoo OpenId no longer exists in social_auth library (PR#1005)
- GitHub backend: Failure with error "No upstream version found" when the project has no new version (#892)
- sar.py fails with AttributeError: 'User' object has no attribute 'social_auth' (#954)

#### Development Changes

- Enhance check_service log output (PR#886)
- Move Anitya from release-monitoring organization to fedora-infra (PR#887)
- Fix documentation build (PR#902)
- Freeze version of dependencies (PR#903)
- Fix service name in vagrant provisioning script (PR#940)
- Add Flask to ReadTheDocs build requirements (PR#946)
- Add pyasn1 to RTD build requirements (PR#947)
- Add support for Python 3.8 (PR#979)
- Make vagrant environment more like production (#924)

#### Other Changes

- Add guidelines for admins on release-monitoring.org (PR#964)
- Add social auth info to SAR script (PR#970)
- Completely remove fedmsg. (#737)
- Add stable_versions field to project.version.update message (#753)
- Fedora messaging schema 2.0.0 - new topic anitya.project.version.update.v2 (#774)
- Rewrite projects pages (#885)
- Update documentation to reflect current state (#972)

#### Contributors

Many thanks to the contributors of bug reports, pull requests, and pull request reviews for this release:

- Anatoli Babenia
- luto
- Michal Konečný
- Olivier Lemasle
- Philippe Ombredanne

### 1.3.2 0.18 (2020-01-13)

#### API Changes

- Filters in APIv2 are now case insensitive (#807)

#### Features

- Automatically delete projects without versions that reached configured error threshold (PR#865)
- GitHub: Store and use latest known version cursors (PR#873)
- Link to commit of latest version if known (PR#874)
- Use dropdown field for distro when on Add project form (#777)
- Add error counter to project (#829)
- Add timeout option for check service (#843)
- Strip whitespaces from version when removing prefix (#866)

#### Bug Fixes

- Fix crash on GNU, Crates and Folder backends (PR#837)
- Fix OOM issue with check_service (PR#842)
- Removed duplicate search form from project search result page (PR#877)
- Fix accessing projects in GitLab subgroups (PR#884)

- Database schema image is missing in documentation (#692)
- Current page is forgotten on login (#713)
- If URL is changed, update ecosystem value as well (#764)
- Tooltips are not working on Firefox 68 (#813)
- Use tag name instead of release name for projects, which are checking only releases (#845)
- Can't disable "Check releases instead of tags" checkbox when editing project (#855)
- Allow no delimiter in calendar versioning pattern (#867)
- Distro search is broken (#876)

#### Development Changes

- Use DEBUG level log for development (PR#826)
- Add Dependabot configuration file (PR#844)
- Bump Vagrant box to Fedora 31 (PR#858)
- Mock the Fedora Messaging calls in the unit tests (PR#860)
- Make *black* show diff of needed changes (PR#878)
- Make log output from check_project_release more readable (#828)

#### Contributors

Many thanks to the contributors of bug reports, pull requests, and pull request reviews for this release:

- Anatoli Babenia
- Aurélien Bompard
- Nicholas La Roux
- Michal Konečný
- Nils Philippsen

### 1.3.3 0.17.2 (2019-09-26)

#### Bug Fixes

- Fix crash on GNU, Crates and Folder backends (PR#837)
- Fix OOM issue with check_service (PR#842)

#### Contributors

Many thanks to the contributors of bug reports, pull requests, and pull request reviews for this release:

- Michal Konečný

### 1.3.4 0.17.1 (2019-09-09)

#### Bug Fixes

- Final info message in check service using error counter instead ratelimit counter (PR#823)
- No error was shown when check_service thread crashed (PR#824)
- Crash when sorting versions with and without date when looking for last retrieved version (PR#825)
- Crash when calling FTP url (PR#833)
- Latest version is not updated when version is removed from web interface (#830)
- GitHub response 403 doesn't have ratelimit reset time (#832)

#### Contributors

Many thanks to the contributors of bug reports, pull requests, and pull request reviews for this release:

- Michal Konečný

### 1.3.5 0.17.0 (2019-09-03)

#### Features

- Adhere to black's Python 3.6 formatting rules (PR#818)
- Support multiple version prefixes (#655)
- Make libraries.io SSE consumer part of Anitya (#723)
- Check for new versions only when there is any change on the URL till last version was retrieved (#730)
- Allow fetching releases on Github backend (#733)
- Add calendar versioning (#740)
- Add semantic versioning (#741)

#### Bug Fixes

- Restore missing closing """ in sample configuration (PR#797)
- Constrain failure during alembic downgrade (PR#812)
- Fix createdb.py to now create all tables properly (PR#817)
- Hide ecosystem field for non admin users (#687)
- Failures during project addition causes distro mapping to be skipped (#735)
- Handle status code 403 as rate limit exception on Github backend (#790)
- Cannot add distro (#791)
- One revision is skipped when doing *alembic upgrade head* (#819)

### Development Changes

- Add docker build to Travis CI tests (PR#799)
- Change required version for pyasn1 (PR#812)
- Minor packaging cleanup and gitignore additions (PR#816)
- Fix rabbitmq-server in dev environment (#804)

### Contributors

Many thanks to the contributors of bug reports, pull requests, and pull request reviews for this release:

- Anatoli Babenia
- Michal Konečný
- Samuel Verschelde
- Vincent Fazio

## 1.3.6 0.16.1 (2019-07-16)

### Bug Fixes

- Check service: Counters saved to database are always 0 (#795)

### Development Changes

- Fix issue with documentation build (#789)

### Contributors

Many thanks to the contributors of bug reports, pull requests, and pull request reviews for this release:

- Michal Konečný

## 1.3.7 0.16.0 (2019-06-24)

### Features

- Turn Anitya cron job to service (#668)

### Bug Fixes

- Error 500 when opening distro page (#709)
- "Edit" form for Distro Mapping forgets the distributions (#744)
- anitya.project.map.new not send when adding new mapping through APIv2 (#760)

#### Development Changes

- Add new dependency ordered_set (#668)
- Add diff-cover to tox testing suite (#782)

#### Contributors

Many thanks to the contributors of bug reports, pull requests, and pull request reviews for this release:

- Michal Konečný

### 1.3.8 0.15.1 (2019-03-06)

#### Bug Fixes

- Fix topic for fedora_messaging (PR#750)

#### Development Changes

- Check formatting using black (PR#725)
- Remove gunicorn dependency (PR#742)

#### Other Changes

- Add sample configuration for Fedora Messaging (#738)

#### Contributors

Many thanks to the contributors of bug reports, pull requests, and pull request reviews for this release:

- Michal Konečný

### 1.3.9 0.15.0 (2019-02-20)

#### Features

- Convert to Fedora Messaging (PR#570)

#### Bug Fixes

- Release notes point to fedora-messaging (#699)
- Javascript error on add project page (#714)
- Changed copyright datum on frontpage to 2013-2019 (#721)
- Invalid User-Agent (#729)

### Development Changes

- Rename Vagrantfile.example to Vagrantfile (PR#715)
- Introduce bandit to tox tests (PR#724)

### Other Changes

- Added example of usage in contribution guide. (PR#719)
- Fix URL to fedmsg website on index.html to use the correct website URL (PR#722)

### Contributors

Many thanks to the contributors of bug reports, pull requests, and pull request reviews for this release:

- Jeremy Cline
- AsciiWolf
- Zlopez
- Michal Konečný
- Neal Gompa
- Yaron Shahrabani

## 1.3.10 0.14.1 (2019-01-17)

### Features

- Show raw version on project page for admins (PR#696)

### Bug Fixes

- Libraries.io consumer is replacing topic_prefix for Anitya (PR#704)
- Release unlocked lock in cronjob (PR#708)
- Comparing by dates created version duplicates (#702)

### Development Changes

- Remove Date version scheme (PR#707)

### Contributors

Many thanks to the contributors of bug reports, pull requests, and pull request reviews for this release:

- Anatoli Babenia
- Michal Konečný

### 1.3.11 0.14.0 (2019-01-08)

#### Features

- Add delete cascade on DB models (PR#608)
- Logs table is replaced by simple status on project (PR#635)
- Update form for adding new distributions (PR#639)
- Refresh page after full check (PR#670)
- Show URL for version check on project UI (#549)
- Link to backend info from project view and edit pages (#556)
- Retrieve all versions, not only the newest one (#595)
- Add rate limit handling (#600)
- Basic user management UI for admins (#621)
- Rate limit enhancements (#665)
- Add ecosystem information to project.version.update fedmsg topic. (#666)

#### Bug Fixes

- Fix unhandled exception in GitLab backend (PR#663)
- Can't rename mapping for gstreamer (#598)
- Source map error: request failed with status 404 for various javascript packages (#606)
- about#test-your-regex link is broken (#628)
- Github backend returns reversed list (#642)
- Version prefix not working in GitLab backend (#644)
- Latest version on Project UI is shown with prefix (#662)
- Crash when version is too long (#674)

#### Development Changes

- Add python 3.7 to tox tests (PR#650)
- Update Vagrantfile to use Fedora 29 image (PR#653)
- Drop support for python 2.7 and python 3.5 (PR#672)

#### Other Changes

- Update contribution guide (PR#636)
- Add GDPR SAR script (PR#649)
- Add supported versions of python to setup script (PR#651)

#### Contributors

Many thanks to the contributors of bug reports, pull requests, and pull request reviews for this release:

- Anatoli Babenia
- Graham Williamson
- Jeremy Cline
- Michal Konečný

### 1.3.12 0.13.2 (2018-10-12)

#### Features

- Show users their ID on Settings page (PR#631)
- Add sorting by creation date for versions (#593)

#### Bug Fixes

- Can't parse owner/repo on Github backend (PR#632)
- Login into staging using OpenID not possible (#616)

#### Development Changes

- Add towncrier for generating release notes (PR#618)
- Remove deprecations warning (PR#627)
- Add documentation dependency to vagrant container (PR#630)

#### Contributors

Many thanks to the contributors of bug reports, pull requests, and pull request reviews for this release:

- Eli Young
- Jeremy Cline
- Michal Konečný

### 1.3.13 v0.13.1

#### Features

- Add database schema generation (#603).

#### Bug Fixes

- Fix cron issues (#613).

### 1.3.14 v0.13.0

#### Dependencies

- Explicitly depend on `defusedxml`

#### Features

- Update GitHub backend to GitHub API v4 (#582).
- Add GitLab backend. This is implemented using GitLab API v4 (#591).
- Update CPAN backend to use metacpan.org (#569).
- Parse XML from CPAN with defusedxml (#569).

#### Bug Fixes

- Change edit message for project, when no edit actually happened (#579).
- Fix wrong title on Edit page (#578).
- Default custom regex is now configurable (#571).

### 1.3.15 v0.12.1

#### Dependencies

- Unpin `straight.plugin` dependency. It was pinned to avoid a bug which has since been fixed in the latest releases (#564).

#### Bug Fixes

- Rather than returning an HTTP 500 when authenticating with two separate identity providers using the same email, return a HTTP 400 to indicate the client should not retry the request and inform them they must log in with the original identity provider (#563).

### 1.3.16 v0.12.0

#### Dependencies

- Drop the dependency on the Python `bunch` package as it is not used.
- There is no longer a hard dependency on the `rpm` Python package.
- Introduce a dependency on the Python `social-auth-app-flask-sqlalchemy` and `flask-login` packages in order to support authenticating against OAuth2, OpenID Connect, and plain OpenID providers.
- Introduce a dependency on the Python `blinker` package to support signaling in Flask.
- Introduce a dependency on the Python `pytoml` package in order to support a TOML configuration format.

### Backwards-incompatible Changes

- Dropped support for Python 2.6
- Added support for Python 3.4+

### APIs

A number of functions that make up Anitya's Python API have been moved (#503). The full list of functions are below. Note that no function signatures have changed.

- `anitya.check_release` is now `anitya.lib.utilities.check_project_release`.
- `anitya.fedmsg_publish` is now `anitya.lib.utilities.fedmsg_publish`.
- `anitya.log` is now `anitya.lib.utilities.log`.
- `anitya.lib.init` is now `anitya.lib.utilities.init`.
- `anitya.lib.create_project` is now `anitya.lib.utilities.create_project`.
- `anitya.lib.edit_project` is now `anitya.lib.utilities.edit_project`.
- `anitya.lib.map_project` is now `anitya.lib.utilities.map_project`.
- `anitya.lib.flag_project` is now `anitya.lib.utilities.flag_project`.
- `anitya.lib.set_flag_state` is now `anitya.lib.utilities.set_flag_state`.
- `anitya.lib.get_last_cron` is now `anitya.lib.utilities.get_last_cron`.

### Deprecations

- Deprecated the v1 HTTP API.

### Features

- Introduced a new set of APIs under `api/v2/` that support write operations for users authenticated with an API token.
- Configuration is now TOML format.
- Added a user guide to the documentation.
- Added an admin guide to the documentation.
- Automatically generate API documentation with Sphinx.
- Introduce httpdomain support to document the HTTP APIs.
- Add initial support for projects to set a "version scheme" in order to help with version ordering. At the present the only version scheme implemented is the RPM scheme.
- Add support for authenticating using a large number of OAuth2, OpenID Connect, and OpenID providers.
- Add a fedmsg consumer that integrates with libraries.io to provide more timely project update notifications.
- Add support for running on OpenShift with s2i.
- Switch over to pypi.org rather than pypi.python.org
- Use HTTPS in backend examples, default URLs, and documentation.

### Bug Fixes

- Fixed deprecation warnings from using `flask.ext` (#431).
- Fix the NPM backend's update feed.

### Developer Improvements

- Fixed all warnings generated from building the Sphinx documentation and introduce tests to ensure there are no regressions (#427).
- Greatly improved the unit tests by breaking monolithic tests up.
- Moved the unit tests into the `anitya.tests` package so tests didn't need to mess with the Python path.
- Fixed logging during test runs
- Switched to pytest as the test runner since nose is dead.
- Introduced nested transactions for database tests rather than removing the database after each test. This greatly reduced run time.
- Added support for testing against multiple Python versions via tox.
- Added Travis CI integration.
- Added code coverage with pytest-cov and Codecov integration.
- Fixed all flake8 errors.
- Refactored the database code to avoid circular dependencies.
- Allow the Vagrant environment to be provisioned with an empty database.

### Contributors

Many thanks to all the contributors for this release, including those who filed issues. Commits for this release were contributed by:

- Elliott Sales de Andrade
- Jeremy Cline
- luto
- Michael Simacek
- Nick Coghlan
- Nicolas Quiniou-Briand
- Ricardo Martincoski
- robled

Thank you all for your hard work.

## 1.3.17 v0.11.0

Released February 08, 2017.

- Return 4XX codes in error cases for /projects/new rather than 200 (Issue #246)

- Allow projects using the "folder" backend to make insecure HTTPS requests (Issue #386)
- Fix an issue where turning the insecure flag on and then off for a project resulted in insecure requests until the server was restarted (Issue #394)
- Add a data migration to set the ecosystem of existing projects if the backend they use is the default backend for an ecosystem. Note that this migration can fail if existing data has duplicate projects since there is a new constraint that a project name is unique within an ecosystem (Issue #402).
- Fix the regular expression used with the Debian backend to strip the "orig" being incorrectly included in the version (Issue #398)
- Added a new backend and ecosystem for https://crates.io (Issue #414)
- [insert summary of change here]

### 1.3.18 v0.10.1

Released November 29, 2016.

- Fix an issue where the version prefix was not being stripped (Issue #372)
- Fix an issue where logs were not viewable to some users (Issue #367)
- Update anitya's mail_logging to be compatible with old and new psutil (Issue #368)
- Improve Anitya's error reporting via email (Issue #368)
- Report the reason fetching a URL failed for the folder backend (Issue #338)
- Add a timeout to HTTP requests Anitya makes to ensure it does not wait indefinitely (Issue #377)
- Fix an issue where prefixes could be stripped further than intended (Issue #381)
- Add page titles to the HTML templates (Issue #371)
- Switch from processes to threads in the Anitya cron job to avoid sharing network sockets for HTTP requests across processes (Issue #335)

## 1.4 Glossary

**ecosystem** Many programming languages now provide a package manager and public collection of packages for that language. Examples include the Python Package Index (PyPI), Rust's Cargo, or JavaScript's NPM.

**backend** A backend in Anitya defines how to retrieve versions in a particular way. For example, some backends might use an API, while others use regular expressions to extract the versions from a web page.

CHAPTER 2

# Admin Guide

## 2.1 Administration Guide

This guide is intended for administrators that are trying to deploy Anitya in their environment. For reference you can check release-monitoring.org ansible deployment role in Fedora infrastructure.

### 2.1.1 Installation

Anitya is not currently available in any distribution's default repository. To install it from PyPI:

```
$ pip install anitya
```

### 2.1.2 Configuration

If the `ANITYA_WEB_CONFIG` environment variable is set to a file system path Anitya can read, the configuration is loaded from that location. Otherwise, the configuration is read from `/etc/anitya/anitya.toml`. If neither can be read, Anitya will log a warning and use its configuration defaults.

> **Warning:** The default configuration for Anitya includes a secret key for web sessions. It is *not* safe to use the default configuration in a production environment.

Anitya uses TOML as its configuration format. A complete configuration file with inline documentation is below.

```
# This is a TOML-format file. For the spec, see https://github.com/toml-lang/toml#spec

# Secret key used to generate the CSRF token in the forms.
secret_key = "changeme please"

# The lifetime of the session, in seconds.
```

(continues on next page)

(continued from previous page)

```
permanent_session_lifetime = 3600

# URL to the database
db_url = "sqlite:////var/tmp/anitya-dev.sqlite"

# List of web administrators. The values should be the value of the "id" column
# for the user in the "users" table of the database. They need to log in before
# this record is created. An example value would be
# "65536ed7-bdd3-4a1e-8252-10d874fd706b"
# You can also find this infromation in the settings page when logged in to Anitya
anitya_web_admins = []

# The email to use in the 'From' header when sending emails.
admin_email = "admin@fedoraproject.org"

# The SMTP server to send mail through
smtp_server = "smtp.example.com"

# Whether or not to send emails to MAIL_ADMIN via SMTP_SERVER when HTTP 500
# errors occur.
email_errors = false

# List of users that are not allowed to sign in, by "id" from the "users" table.
blacklisted_users = []

# The type of session protection used by social-auth.
session_protection = "strong"

# The authentication backends to use. For valid values, see social-auth's
# documentation.
social_auth_authentication_backends = [
    "social_core.backends.fedora.FedoraOpenId",
    "social_core.backends.gitlab.GitLabOAuth2",
    "social_core.backends.github.GithubOAuth2",
    "social_core.backends.google.GoogleOAuth2",
    "social_core.backends.yahoo.YahooOAuth2",
    "social_core.backends.open_id.OpenIdAuth"
]

# Force the application to require HTTPS on authentication redirects.
social_auth_redirect_is_https = true

# List of platforms for which Anitya should automatically create new projects
# when Libraries.io announces a new version. See https://libraries.io/ for the
# list of valid platforms. By default, Anitya will only update existing projects
# via Libraries.io.
librariesio_platform_whitelist = []
sse_feed = "http://firehose.libraries.io/events"

# Default regular expression used for backend
default_regex = """\
                (?i)%(name)s(?:[-_]?(?:minsrc|src|source))?[-_]([^-/_\\s]+?)(?:[-_]\
                (?:minsrc|src|source|asc|release))?\\.(?:tar|t[bglx]z|tbz2|zip)\
                """

# Github access token
# This is used by GitHub API for github backend
```

(continues on next page)

(continued from previous page)

```
# Permission needed by Anitya:
# * repo:status
# * public_repo
github_access_token = "foobar"

# Check service configuration
# Number of workers
cron_pool = 10
# Worker timeout in seconds
check_timeout = 600
# When this number of failed checks is reached,
# project will be automatically removed, if no version was retrieved yet
check_error_threshold=100

# The logging configuration, in Python dictConfig format.
[anitya_log_config]
    version = 1
    disable_existing_loggers = true

    [anitya_log_config.formatters]
        [anitya_log_config.formatters.simple]
            format = "[%(name)s %(levelname)s] %(message)s"

    [anitya_log_config.handlers]
        [anitya_log_config.handlers.console]
            class = "logging.StreamHandler"
            formatter = "simple"
            stream = "ext://sys.stdout"

    [anitya_log_config.loggers]
        [anitya_log_config.loggers.anitya]
            level = "WARNING"
            propagate = false
            handlers = ["console"]

    [anitya_log_config.root]
        level = "ERROR"
        handlers = ["console"]
```

Anitya uses second configuration file for Fedora messaging. A sample configuration file is bellow. To know more about the configuration of fedora messaging please refer to fedora messaging configuration documentation.

```
# A sample configuration for fedora-messaging. This file is in the TOML format.
# For complete details on all configuration options, see the documentation.
# https://fedora-messaging.readthedocs.io/en/latest/configuration.html

amqp_url = "amqp://"

[tls]
ca_cert = "/etc/pki/tls/certs/ca-bundle.crt"
keyfile = "/my/client/key.pem"
certfile = "/my/client/cert.pem"
```

### 2.1.3 Services

Anitya is made up of a *WSGI Application*, an *Update Service* that could be run separately, an optional *Libraries.io SSE client*, *SAR Script*, and requires a *Database*.

#### WSGI Application

The WSGI application is located at `anitya/wsgi.py`. This application handles the web interface for creating, updating, and viewing projects. It also offers a REST API.

There is also a `anitya.wsgi` file that could be used directly. You can find example anitya.wsgi in Anitya repository. You can use this file with Apache server or deploy it by flask. Fedora uses Apache so you can look at their configuration.

#### Update Service

The service that checks for project updates is located at `anitya/check_service.py` in the git repository and Python package. To enable it, just start this service.

---

**Note:** This script should be also available system wide, installed by `` `scripts `` argument in python setup. See python setup documentation for more info.

---

#### Libraries.io SSE client

This optional service listens to SSE feed for messages published by the libraries.io service.

The service is located at `anitya/librariesio_consumer.py`. To enable it just start the service.

---

**Note:** This script should be also available system wide, installed by `` `scripts `` argument in python setup. See python setup documentation for more info.

---

#### SAR Script

Subject Access Requests script is intended for handling GDPR users requests for obtaining their data from Anitya. This script could be found in `anitya/sar.py`. It just connects to the database using Anitya configuration and takes out user relevant data.

---

**Note:** This script should be also available system wide, installed by `` `scripts `` argument in python setup. See python setup documentation for more info.

---

#### Database

Anitya should work with any SQL database, but it is only tested with SQLite and PostgreSQL. It is recommended to use PostgreSQL in a production deployment. The SQLite database can't work with update service, because it doesn't allow database changes in parallel threads.

For creating a database schema you can use createdb.py script from Anitya repository.

After this you need to apply any migrations done above the basic database schema. You can run the migrations by using the alembic tool. You can use the configuration file alembic.ini from Anitya repository.

```
alembic -c <path_to_alembic.ini> upgrade head
```

**Note:** The migrations needs to be applied each time upgrade of Anitya is done.

#### Fedora messaging

The Anitya needs to connect to RabbitMQ server which will listen for it's messages. For deployment of your own RabbitMQ server please look at the official documentation.

## 2.2 Integrating with Anitya

This chapter describes ways how you can integrate Anitya with your solution.

### 2.2.1 Fedora messaging

Fedora messaging is a message bus. In other words it is a system that allows for the sending and receiving of notifications between applications. For anitya, every action made on the application is announced/broadcasted on this bus, allowing anyone listening to it to act immediately instead of (for example) polling hourly all the data, looking for changes, and acting then.

To start receiving Fedora messaging messages from anitya, it is as simple as:

- install `Fedora messaging` the way you want:

On Fedora

```
dnf install fedora-messaging
```

Via pip

```
pip install fedora-messaging
```

For how to start a local broker for *fedora messaging*. See Fedora messaging documentation.

#### List of fedora messaging topics

This section will list all the Fedora messaging topics Anitya is sending and in what situation.

- *anitya.distro.add* is sent when a new distribution is created.
- *anitya.distro.edit* is sent when an existing distribution is edited.
- *anitya.distro.remove* is sent when an existing distribution is deleted.
- *anitya.project.add* is sent when a new project is added.
- *anitya.project.edit* is sent when an existing project is edited.
- *anitya.project.remove* is sent when an existing project is deleted.
- *anitya.project.flag* is sent when a new flag is created on project.

- *anitya.project.flag.set* is sent when a status of flag is changed.
- *anitya.project.map.new* is sent when a new mapping to distribution is created on project.
- *anitya.project.map.update* is sent when an existing mapping on project is edited.
- *anitya.project.map.remove* is sent when an existing mapping on project is deleted.
- *anitya.project.version.update* is sent when a new release is found for the project. This topic is now deprecated.
- *anitya.project.version.update.v2* is sent when a new release is found for the project. This message differentiate between stable and not stable releases.
- *anitya.project.version.remove* is sent when an existing release is deleted from project.

You can found out more about Anitya messages in Fedora messaging schema. The schema should be used by every consumer that consumes Anitya messages.

CHAPTER 3

# Developer Guide

## 3.1 API Documentation

Anitya provides several APIs for users.

### 3.1.1 HTTP API v2

The token for API could be obtained from settings page in Anitya web interface. This token needs to be provided in `Authorization` header of the request. See request examples bellow.

**`POST /api/v2/packages/`**
Create a new package associated with an existing project and distribution.

**Example request**:

```
POST /api/v2/packages/ HTTP/1.1
Accept: application/json
Accept-Encoding: gzip, deflate
Authorization: Token gAOFi2wQPzUJFIfDkscAKjbJfXELCz0r44m57Ur2
Connection: keep-alive
Content-Length: 120
Content-Type: application/json
Host: localhost:5000
User-Agent: HTTPie/0.9.4

{
    "distribution": "Fedora",
    "package_name": "python-requests",
    "project_ecosystem": "pypi",
    "project_name": "requests"
}
```

```
HTTP/1.0 201 CREATED
Content-Length: 69
Content-Type: application/json
Date: Mon, 15 Jan 2018 21:49:01 GMT
Server: Werkzeug/0.14.1 Python/2.7.14

{
    "distribution": "Fedora",
    "name": "python-requests"
}
```

**Request Headers**

- Authorization – API token to use for authentication

**Request JSON Object**

- **distribution** (*string*) – The name of the distribution that contains this package.
- **package_name** (*string*) – The name of the package in the distribution repository.
- **project_name** (*string*) – The project name in Anitya.
- **project_ecosystem** (*string*) – The ecosystem the project is a part of. If it's not part of an ecosystem, use the homepage used in the Anitya project.

**Status Codes**

- 201 Created – When the package was successfully created.
- 400 Bad Request – When required arguments are missing or malformed.
- 401 Unauthorized – When your access token is missing or invalid
- 409 Conflict – When the package already exists.

**GET /api/v2/packages/**

List all packages.

**Example request**:

```
GET /api/v2/packages/?name=0ad&distribution=Fedora HTTP/1.1
Accept: application/json
Accept-Encoding: gzip, deflate
Connection: keep-alive
Host: localhost:5000
User-Agent: HTTPie/0.9.4
```

**Example response**:

```
HTTP/1.0 200 OK
Content-Length: 181
Content-Type: application/json
Date: Mon, 15 Jan 2018 20:21:44 GMT
Server: Werkzeug/0.14.1 Python/2.7.14

{
    "items": [
        {
            "distribution": "Fedora",
            "name": "python-requests"
```

(continues on next page)

(continued from previous page)

```
            "project": "requests",
            "ecosystem": "pypi",
        }
    ],
    "items_per_page": 25,
    "page": 1,
    "total_items": 1
}
```

**Query Parameters**

- **page** (*int*) – The package page number to retrieve (defaults to 1).
- **items_per_page** (*int*) – The number of items per page (defaults to 25, maximum of 250).
- **distribution** (*str*) – Filter packages by distribution.
- **name** (*str*) – The name of the package.

**Status Codes**

- 200 OK – If all arguments are valid. Note that even if there are no projects, this will return 200.
- 400 Bad Request – If one or more of the query arguments is invalid.

**POST /api/v2/projects/**

Create a new project.

**Example Request**:

```
POST /api/v2/projects/ HTTP/1.1
Authorization: token hxPpKow7nnT6UTAEKMtQwl310P6GtyqV8DDbexnk
Accept: application/json
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Length: 114
Content-Type: application/json
Host: localhost:5000
User-Agent: HTTPie/0.9.4

{
    "backend": "custom",
    "homepage": "https://example.com/test",
    "name": "test_project",
    "version_prefix": "release-"
}
```

**Example Response**:

```
HTTP/1.0 201 CREATED
Content-Length: 276
Content-Type: application/json
Date: Sun, 26 Mar 2017 15:56:30 GMT
Server: Werkzeug/0.12.1 Python/2.7.13

{
```

(continues on next page)

(continued from previous page)


```
    "backend": "PyPI",
    "created_on": 1490543790.0,
    "homepage": "http://python-requests.org",
    "id": 13857,
    "name": "requests",
    "regex": null,
    "updated_on": 1490543790.0,
    "version": null,
    "version_url": null,
    "versions": []
}
```


**Query Parameters**

- **access_token** (*string*) – Your API access token.

**Request JSON Object**

- **name** (*string*) – The project name
- **homepage** (*string*) – The project homepage URL
- **backend** (*string*) – The project backend (github, folder, etc.).
- **version_url** (*string*) – The URL to fetch when determining the project version (defaults to null).
- **version_prefix** (*string*) – The project version prefix, if any. For example, some projects prefix with "v".
- **regex** (*string*) – The regex to use when searching the `version_url` page.
- **insecure** (*bool*) – When retrieving the versions via HTTPS, do not validate the certificate (defaults to false).
- **check_release** (*bool*) – Check the release immediately after creating the project.

**Status Codes**

- 201 Created – When the project was successfully created.
- 400 Bad Request – When required arguments are missing or malformed.
- 401 Unauthorized – When your access token is missing or invalid, or when the server is not configured for OpenID Connect. The response will include a JSON body describing the exact problem.
- 409 Conflict – When the project already exists.

**GET /api/v2/projects/**

Lists all projects.

**Example request**:

```
GET /api/v2/projects/?items_per_page=1&page=2 HTTP/1.1
Accept: application/json
Accept-Encoding: gzip, deflate
Connection: keep-alive
Host: localhost:5000
User-Agent: HTTPie/0.9.4
```

**Example response**:

```
HTTP/1.0 200 OK
Content-Length: 676
Content-Type: application/json
Date: Fri, 24 Mar 2017 18:44:32 GMT
Server: Werkzeug/0.12.1 Python/2.7.13

{
    "items": [
        {
            "backend": "Sourceforge",
            "created_on": 1412174943.0,
            "ecosystem": "https://sourceforge.net/projects/zero-install",
            "homepage": "https://sourceforge.net/projects/zero-install",
            "id": 1,
            "name": "0install",
            "regex": "",
            "updated_on": 1482495004.0,
            "version": "2.12",
            "version_url": "zero-install",
            "versions": [
                "2.12",
                "2.11",
                "2.10",
                "2.9.1",
                "2.9",
                "2.8",
                "2.7"
            ]
        }
    ],
    "items_per_page": 1,
    "page": 2,
    "total_items": 13468
}
```

**Query Parameters**

- **page** (*int*) – The project page number to retrieve (defaults to 1).
- **items_per_page** (*int*) – The number of items per page (defaults to 25, maximum of 250).
- **ecosystem** (*string*) – The project ecosystem (e.g. pypi, rubygems). If the project is not part of a language package index, use its homepage.
- **name** (*string*) – The project name to filter the query by.

**Status Codes**

- 200 OK – If all arguments are valid. Note that even if there are no projects matching the query, this will return 200.
- 400 Bad Request – If one or more of the query arguments is invalid.

**POST /api/v2/versions/**

Check for new versions on the project. The API first checks if the project exists, if exists it will do check on it while applying any changes. If not it will create a temporary object in memory and do a check on the temporary object.

**Example Request**:

```
POST /api/v2/versions/ HTTP/1.1
Accept: application/json, */*
Accept-Encoding: gzip, deflate
Authorization: token s12p01zUiVdEOZIhVf0jyZqtyYXfo2DECi6YdqqV
Connection: keep-alive
Content-Length: 15
Content-Type: application/json
Host: localhost:5000
User-Agent: HTTPie/1.0.3

{
    "id": "55612"
}
```

**Example Response**:

```
HTTP/1.0 200 OK
Content-Length: 118
Content-Type: application/json
Date: Tue, 20 Oct 2020 08:49:01 GMT
Server: Werkzeug/0.16.0 Python/3.8.6

{
    "found_versions": [],
    "latest_version": "0.0.2",
    "versions": [
        "0.0.2",
        "0.0.1"
    ]
}
```

**Query Parameters**

- **access_token** (*string*) – Your API access token.

**Request JSON Object**

- **id** (*int*) – Id of the project. If provided the check is done above existing project.
- **name** (*string*) – The project name. Used as a filter to find existing project, if id not provided.
- **homepage** (*string*) – The project homepage URL. Used as a filter to find existing project, if id not provided.
- **backend** (*string*) – The project backend (github, folder, etc.).
- **version_url** (*string*) – The URL to fetch when determining the project version.
- **version_scheme** (*string*) – The project version scheme (defaults to "RPM" for temporary project).
- **version_pattern** (*string*) – The version pattern for calendar version scheme.
- **version_prefix** (*string*) – The project version prefix, if any.
- **pre_release_filter** (*string*) – Filter for unstable versions.
- **version_filter** (*string*) – Filter for blacklisted versions.
- **regex** (*string*) – The regex to use when searching the `version_url` page (defaults to none for temporary project).

- **insecure** (*bool*) – When retrieving the versions via HTTPS, do not validate the certificate (defaults to false for temporary project).
- **releases_only** (*bool*) – When retrieving the versions, use releases instead of tags (defaults to false for temporary project). Only available for GitHub backend.
- **dry_run** (*bool*) – If set, doesn't save anything (defaults to true). Can't be set to False for temporary project.

**Status Codes**

- 200 OK – When the versions were successfully retrieved.
- 400 Bad Request – When required arguments are missing or malformed.
- 401 Unauthorized – When your access token is missing or invalid, or when the server is not configured for OpenID Connect. The response will include a JSON body describing the exact problem.
- 404 Not Found – When id is provided and the project doesn't exist or is archived.
- 500 Internal Server Error – If there is error during the check

**GET /api/v2/versions/**

Lists all versions on project.

**Example request**:

```
GET /api/v2/versions/?project_id=1 HTTP/1.1
Accept: application/json
Accept-Encoding: gzip, deflate
Connection: keep-alive
Host: localhost:5000
User-Agent: HTTPie/0.9.4
```

**Example response**:

```
HTTP/1.0 200 OK
Content-Length: 676
Content-Type: application/json
Date: Fri, 24 Mar 2017 18:44:32 GMT
Server: Werkzeug/0.12.1 Python/2.7.13

{
    "latest_version": "2.12",
    "versions": [
        "2.12",
        "2.11",
        "2.10",
        "2.9.1",
        "2.9",
        "2.8",
        "2.7"
    ]
}
```

**Query Parameters**

- **project_id** (*int*) – The id of the project we want to get versions for.

**Status Codes**

- 200 OK – If all arguments are valid.
- 400 Bad Request – If one or more of the query arguments is invalid.
- 404 Not Found – If project with specified id doesn't exist.

### 3.1.2 HTTP API v1

**GET /api**

**GET /api/**
Retrieve the HTML information page.

**Deprecated** in Anitya 0.12 in favor of simple Sphinx documentation.

**Status Codes**

- 302 Found – A redirect to the HTML documentation.

**GET /api/by_ecosystem/** (*ecosystem*) **/**
*project_name*

**GET /api/by_ecosystem/** (*ecosystem*) **/**
*project_name***/** Retrieves a project in an ecosystem via the name of the ecosystem and the name of the project as registered with Anitya.

**Parameters**

- **ecosystem** (*str*) – the name of the ecosystem (case insensitive).
- **project_name** (*str*) – the name of the project in Anitya.

**Status Codes**

- 200 OK – Returns the JSON representation of the project.
- 404 Not Found – When either the ecosystem does not exist, or when there is no project with that name within the ecosystem.

**Example request**:

```
GET /api/by_ecosystem/pypi/six HTTP/1.1
Accept: application/json
Accept-Encoding: gzip, deflate
Connection: keep-alive
Host: release-monitoring.org
User-Agent: HTTPie/0.9.4
```

**Example response**:

```
HTTP/1.1 200 OK
Content-Length: 516
Content-Type: application/json

{
  "backend": "pypi",
  "created_on": 1409917222.0,
  "homepage": "https://pypi.python.org/pypi/six",
  "id": 2,
  "name": "six",
  "packages": [
    {
```

(continues on next page)

(continued from previous page)

```
      "distro": "Fedora",
      "package_name": "python-six"
    }
  ],
  "regex": null,
  "updated_on": 1414400794.0,
  "version": "1.10.0",
  "version_url": null,
  "versions": [
    "1.10.0"
  ]
}
```

**GET /api/distro/names**

**GET /api/distro/names/**

Lists the names of all the distributions registered in anitya.

**Query Parameters**

- **pattern** – pattern to use to restrict the list of distributions returned.

**Example request**:

```
GET /api/distro/names/?pattern=F* HTTP/1.1
Accept: application/json
Accept-Encoding: gzip, deflate
Connection: keep-alive
Host: release-monitoring.org
User-Agent: HTTPie/0.9.4
```

**Example response**:

```
HTTP/1.1 200 OK
Content-Length: 79
Content-Type: application/json

{
    "distro": [
        "Fedora",
        "Fedora EPEL",
        "FZUG"
    ],
    "total": 3
}
```

**GET /api/packages/wiki**

**GET /api/packages/wiki/**

List all packages in mediawiki format.

**Deprecated** in Anitya 0.12 due to lack of pagination resulting in incredibly poor performance.

Lists all the packages registered in anitya using the format of the old wiki page. If a project is present multiple times on different distribution, it will be shown multiple times.

```
/api/packages/wiki
```

Accepts GET queries only.

Sample response:

```
* 2ping None https://www.finnie.org/software/2ping
* 3proxy None https://www.3proxy.ru/download/
```

**GET /api/project/**(*distro*)**/**
**path:** *package_name*

**GET /api/project/**(*distro*)**/**
**path:** *package_name***/** Retrieves a project in a distribution via the name of the distribution and the name of the package in said distribution.

```
/api/project/<distro>/<package_name>
```

Accepts GET queries only.

**Parameters**

- **distro** – the name of the distribution (case insensitive).
- **package_name** – the name of the package in the distribution specified.

Sample response:

```
{
  "backend": "custom",
  "created_on": 1409917222.0,
  "homepage": "https://www.finnie.org/software/2ping/",
  "id": 2,
  "name": "2ping",
  "packages": [
    {
      "distro": "Fedora",
      "package_name": "2ping"
    }
  ],
  "regex": null,
  "updated_on": 1414400794.0,
  "version": "2.1.1",
  "version_url": "https://www.finnie.org/software/2ping",
  "versions": [
    "2.1.1"
  ]
}
```

**GET /api/project/**(**int:** *project_id*)

**GET /api/project/**(**int:** *project_id*)**/**
Retrieves a specific project using its identifier in anitya.

```
/api/project/<project_id>
```

Accepts GET queries only.

**Parameters**

- **project_id** – the identifier of the project in anitya.

Sample response:

```
{
  "backend": "custom",
  "created_on": 1409917222.0,
  "homepage": "https://www.finnie.org/software/2ping/",
  "id": 2,
  "name": "2ping",
  "packages": [
    {
      "distro": "Fedora",
      "package_name": "2ping"
    }
  ],
  "regex": null,
  "updated_on": 1414400794.0,
  "version": "2.1.1",
  "version_url": "https://www.finnie.org/software/2ping",
  "versions": [
    "2.1.1"
  ]
}
```


**GET /api/projects**

**GET /api/projects/**

Lists all the projects registered in Anitya.

This API accepts GET query strings:

```
/api/projects

/api/projects/?pattern=<pattern>

/api/projects/?pattern=py*

/api/projects/?homepage=<homepage>

/api/projects/?homepage=https%3A%2F%2Fpypi.python.org%2Fpypi%2Fansi2html
```

Accepts GET queries only.

**Kwarg pattern** pattern to use to restrict the list of projects returned.

**Kwarg homepage** upstream homepage to use to restrict the list of projects returned.

The `pattern` and `homepage` arguments are mutually exclusive and cannot be combined. You can query for packages by a pattern **or** you can query by their upstream homepage, but not both.

Sample response:


```
{
  "projects": [
    {
      "backend": "custom",
      "created_on": 1409917222.0,
      "homepage": "https://www.finnie.org/software/2ping/",
      "id": 2,
      "name": "2ping",
      "regex": null,
      "updated_on": 1414400794.0,
```


(continues on next page)

(continued from previous page)


```
      "version": "2.1.1",
      "version_url": "https://www.finnie.org/software/2ping",
      "versions": [
        "2.1.1"
      ]
    },
    {
      "backend": "custom",
      "created_on": 1409917223.0,
      "homepage": "https://www.3proxy.ru/download/",
      "id": 3,
      "name": "3proxy",
      "regex": null,
      "updated_on": 1415115096.0,
      "version": "0.7.1.1",
      "version_url": "https://www.3proxy.ru/download/",
      "versions": [
        "0.7.1.1"
      ]
    }
  ],
  "total": 2
}
```


**GET /api/projects/names**

**GET /api/projects/names/**

Lists the names of all the projects registered in anitya.

**Query Parameters**

- **pattern** (*str*) – pattern to use to restrict the list of names returned.

**Status Codes**

- 200 OK – Returned in all cases.

**Example request**:

```
GET /api/projects/names?pattern=requests* HTTP/1.1
Accept: application/json
Accept-Encoding: gzip, deflate
Connection: keep-alive
Host: release-monitoring.org
User-Agent: HTTPie/0.9.4
```

**Example response**:


```
HTTP/1.1 200 OK
Content-Length: 248
Content-Type: application/json

{
    "projects": [
        "requests",
        "Requests",
        "requests-aws",
        "requestsexceptions",
        "requests-file",
```


(continues on next page)

(continued from previous page)

```
        "requests-ftp",
        "requests-mock",
        "requests-ntlm",
        "requests-oauthlib",
        "requests-toolbelt"
    ],
    "total": 10
}
```

**GET /api/version**

**GET /api/version/**
Display the api version information.

```
/api/version
```

Accepts GET queries only.

Sample response:

```
{
  "version": "1.0"
}
```

**POST /api/version/get**
Forces anitya to retrieve new versions available from a project upstream.

```
/api/version/get
```

Accepts POST queries only.

**Parameters**

- **id** – the identifier of the project in anitya.

Sample response:

```
{
  "versions": [
    "2.7"
  ]
}
```

Sample error response:

```
{
  "output": "notok",
  "error": "Error happened."
}
```

### 3.1.3 Python APIs

#### Exceptions

Exceptions used by Anitya.

**Authors:** Pierre-Yves Chibon <pingou@pingoured.fr>

**exception** anitya.lib.exceptions.**AnityaException**
Bases: Exception

Generic class covering all the exceptions generated by anitya.

**exception** anitya.lib.exceptions.**AnityaInvalidMappingException**(*pkgname*, *distro*, *found_pkgname*, *found_distro*, *project_id*, *project_name*, *link=None*)
Bases: *anitya.lib.exceptions.AnityaException*

Specific exception class for invalid mapping.

**message**

**exception** anitya.lib.exceptions.**AnityaPluginException**
Bases: *anitya.lib.exceptions.AnityaException*

Generic exception class that can be used by the plugin to indicate an error.

**exception** anitya.lib.exceptions.**InvalidVersion**(*version*, *exception=None*)
Bases: *anitya.lib.exceptions.AnityaException*

Raised when the version string is not valid for the given version scheme.

**Parameters**

- **version** (*str*) – The version string that failed to parse.
- **exception** (*Exception*) – The underlying exception that triggered this one.

**exception** anitya.lib.exceptions.**ProjectExists**(*requested_project*)
Bases: *anitya.lib.exceptions.AnityaException*

Raised when a project already exists in the database.

This is only raised when a project is part of an ecosystem, since projects outside of an ecosystem have no uniqueness constraints.

**to_dict**()

**exception** anitya.lib.exceptions.**RateLimitException**(*reset_time*)
Bases: *anitya.lib.exceptions.AnityaException*

Raised when the rate limit for requests is reached.

**reset_time**
Time when limit will be reset.

**Type** *arrow.Arrow*

### Database API

This package contains all the database-related code, including SQLAlchemy models, Alembic migrations, and a scoped session object configured from anitya.config

### anitya.db.meta

This module sets up the basic database objects that all our other modules will rely on. This includes the declarative base class and global scoped session.

This is in its own module to avoid circular imports from forming. Models and events need to be imported by `__init__.py`, but they also need access to the *Base* model and *Session*.

**class** anitya.db.meta.**Base**(***kwargs*)
Bases: anitya.db.meta._AnityaBase

Base class for the SQLAlchemy model base class.

**query**
a class property which produces a *BaseQuery* object against the class and the current Session when called. Classes that want a customized Query class should sub-class *BaseQuery* and explicitly set the query property on the model.

**Type** sqlalchemy.orm.query.Query

**metadata = MetaData()**

**registry = <sqlalchemy.orm.decl_api.registry object>**

**class** anitya.db.meta.**BaseQuery**(*entities*, *session=None*)
Bases: sqlalchemy.orm.query.Query

A base Query object that provides queries.

**paginate**(*page=None*, *items_per_page=None*, *order_by=None*)
Retrieve a page of items.

**Parameters**

- **page** (*int*) – the page number to retrieve. This page is 1-indexed and defaults to 1.
- **items_per_page** (*int*) – The number of items per page. This defaults to 25.
- **order_by** (*sa.Column or tuple*) – One or more criterion by which to order the pages.

**Returns** A namedtuple of the items.

**Return type** *Page*

**Raises** ValueError – If the page or items_per_page values are less than 1.

**class** anitya.db.meta.**Page**
Bases: anitya.db.meta._Page

A sub-class of namedtuple that represents a page.

**items**
The database objects from the query.

**Type** object

**page**
The page number used for the query.

**Type** int

**items_per_page**
The number of items per page.

**Type** int

**total_items**
The total number of items in the database.

**Type** int

**as_dict**()
Return a dictionary representing the page.

**Returns**

**A dictionary representation of the page and its items, using** the __json__ method defined on the item objects.

**Return type** dict

anitya.db.meta.**Session = <sqlalchemy.orm.scoping.scoped_session object>**
This is a configured scoped session. It creates thread-local sessions. This means that Session() is Session() is True. This is a convenient way to avoid passing a session instance around. Consult SQLAlchemy's documentation for details.

Before you can use this, you must call *initialize()*.

anitya.db.meta.**initialize**(*config*)
Initialize the database.

This creates a database engine from the provided configuration and configures the scoped session to use the engine.

**Parameters** **config** (*dict*) – A dictionary that contains the configuration necessary to initialize the database.

**Returns** The database engine created from the configuration.

**Return type** sqlalchemy.engine

### anitya.db.events

This module contains functions that are triggered by SQLAlchemy events.

anitya.db.events.**set_ecosystem_backend**(*target*, *value*, *old*, *initiator*)
An SQLAlchemy event listener that sets the ecosystem for a project if backend is changed.

**Parameters**

- **target** (*sqlalchemy.orm.state.InstanceStace*) – Instance of the object where change is happening.
- **value** (*str*) – The new value of backend.
- **old** (*str*) – The old value of backend.
- **initiator** (*sqlalchemy.orm.attributes.Event*) – The event object that is initiating this transition.

anitya.db.events.**set_ecosystem_homepage**(*target*, *value*, *old*, *initiator*)
An SQLAlchemy event listener that sets the ecosystem for a project if homepage is changed.

**Parameters**

- **target** (*sqlalchemy.orm.state.InstanceStace*) – Instance of the object where change is happening.
- **value** (*str*) – The new value of homepage.
- **old** (*str*) – The old value of homepage.
- **initiator** (*sqlalchemy.orm.attributes.Event*) – The event object that is initiating this transition.

### anitya.db.models

SQLAlchemy database models.

**class** anitya.db.models.**ApiToken**(***kwargs*)
Bases: sqlalchemy.orm.decl_api.Base

A table for user API tokens.

**token**
A 40 character string that represents the API token. This is the primary key and is, by default, generated automatically.

**Type** sa.String

**created**
The time this API token was created.

**Type** sa.DateTime

**description**
A user-provided description of what the API token is for.

**Type** sa.Text

**user**
The user this API token is associated with.

**Type** User

**created**

**description**

**token**

**user**

**user_id**

**class** anitya.db.models.**Distro**(*name*)
Bases: sqlalchemy.orm.decl_api.Base

**classmethod all**(*session*, *page=None*, *count=False*)

**classmethod by_name**(*session*, *name*)

**classmethod get**(*session*, *name*)

**classmethod get_or_create**(*session*, *name*)

**name**

**package**

**classmethod search**(*session*, *pattern*, *page=None*, *count=False*)
Search the distribuutions by their name

**class** anitya.db.models.**GUID**(**args*, ***kwargs*)
Bases: sqlalchemy.sql.type_api.TypeDecorator

Platform-independent GUID type.

If PostgreSQL is being used, use its native UUID type, otherwise use a CHAR(32) type.

**impl**
alias of sqlalchemy.sql.sqltypes.CHAR

**load_dialect_impl**(*dialect*)
PostgreSQL has a native UUID type, so use it if we're using PostgreSQL.

**Parameters** **dialect** (*sqlalchemy.engine.interfaces.Dialect*) – The dialect in use.

**Returns**

**Either a PostgreSQL UUID or a CHAR(32) on other** dialects.

**Return type** sqlalchemy.types.TypeEngine

**process_bind_param**(*value*, *dialect*)
Process the value being bound.

If PostgreSQL is in use, just use the string representation of the UUID. Otherwise, use the integer as a hex-encoded string.

**Parameters**

- **value** (*object*) – The value that's being bound to the object.
- **dialect** (*sqlalchemy.engine.interfaces.Dialect*) – The dialect in use.

**Returns** The value of the UUID as a string.

**Return type** str

**process_result_value**(*value*, *dialect*)
Casts the UUID value to the native Python type.

**Parameters**

- **value** (*object*) – The database value.
- **dialect** (*sqlalchemy.engine.interfaces.Dialect*) – The dialect in use.

**Returns** The value as a Python `uuid.UUID`.

**Return type** uuid.UUID

**class** anitya.db.models.**Packages**(***kwargs*)
Bases: sqlalchemy.orm.decl_api.Base

**classmethod by_id**(*session*, *pkg_id*)

**classmethod by_package_name_distro**(*session*, *package_name*, *distro_name*)

**distro**

**distro_name**

**classmethod get**(*session*, *project_id*, *distro_name*, *package_name*)

**id**

**package_name**

**project**

**project_id**

**class** anitya.db.models.**Project**(***kwargs*)
Bases: sqlalchemy.orm.decl_api.Base

Models an upstream project and maps it to a database table.

**id**
The database primary key.

**Type** sa.Integer

**name**
The upstream project's name.

**Type** sa.String

**homepage**
The URL for the project's home page.

**Type** sa.String

**backend**
The name of the backend to use when fetching updates; this is a foreign key to a `Backend`.

**Type** sa.String

**ecosystem_name**
The name of the ecosystem this project is a part of. If the project isn't part of an ecosystem (e.g. PyPI), use the homepage URL.

**Type** sa.String

**version_url**
The url to use when polling for new versions. This may be ignored if this project is part of an ecosystem with a fixed URL (e.g. Cargo projects are on https://crates.io).

**Type** sa.String

**regex**
A Python `re` style regular expression that is applied to the HTML from `version_url` to find versions.

**Type** sa.String

**version_prefix**
A string containing version prefixes delimited by ';'. These prefixes will be removed from string showed on page.

**Type** sa.String

**version_pattern**
A version pattern used for calendar version scheme.

**Type** sa.String

**insecure**
Whether or not to validate the x509 certificate offered by the server at `version_url`. Defaults to `False`.

**Type** sa.Boolean

**releases_only**
Whether or not to check releases instead of tags. This is now only used by GitHub backend.

**Type** sa.Boolean

**error_counter**
Counter that contains number of unsuccessful checks. This counter will reset each time a successful check is done. Doesn't count ratelimit errors.

**Type** sa.Integer

**latest_version**
The latest version for the project, as determined by the version sorting algorithm.

**Type** sa.String

**latest_version_cursor**
A backend-specific cursor to the latest version for the project, as determined by the version sorting algorithm. This can be used to avoid paginating over every old version.

**Type** sa.String

**logs**
The result of the last update.

**Type** sa.Text

**check_successful**
Flag that contains result of last check. `None` - not checked yet, `True` - checked successfully, `False` - error occured during check

**Type** sa.Boolean

**updated_on**
When the project was last updated.

**Type** sa.DateTime

**created_on**
When the project was created in Anitya.

**Type** sa.DateTime

**packages**
List of `Package` objects which represent the downstream packages for this project.

**Type** list

**version_scheme**
The version scheme to use for this project. If this is null, a default will be used. See the `anitya.lib.versions` documentation for more information.

**Type** sa.String

**pre_release_filter**
A string containing filters delimited by ';'. Filtered versions will be marked as pre_release.

**Type** sa.String

**archived**
Marks the project as archived, archived projects can't be edited by normal users and are no longer checked for new versions.

**Type** sa.Boolean

**version_filter**
A string containing filters delimited by ';'. Filtered versions will be skipped when retrieving versions.

**Type** sa.String

**classmethod all**(*session*, *page=None*, *count=False*)

**archived**

**backend**

**classmethod by_distro**(*session*, *distro*, *page=None*, *count=False*)

**classmethod by_homepage**(*session*, *homepage*)

**classmethod by_id**(*session*, *project_id*)

**classmethod by_name**(*session*, *name*)

**classmethod by_name_and_ecosystem**(*session*, *name*, *ecosystem*)

**classmethod by_name_and_homepage**(*session*, *name*, *homepage*)

**check_successful**

**create_version_objects**(*versions*)
Creates sorted list of version objects defined by *self.version_class* from versions list.

**Parameters** **versions** (*list(str or dict)*) – List of versions that are not associated with the project.

**Returns**

**List of version objects defined by** *self.version_class*.

**Return type** list(*anitya.lib.versions.Base*)

**created_on**

**ecosystem_name**

**error_counter**

**flags**

**classmethod get**(*session*, *project_id*)

**get_last_created_version**()
Returns last obtained release by date.

**Returns** Version object or None, if project doesn't have any version yet.

**Return type** (*ProjectVersion*)

**classmethod get_or_create**(*session*, *name*, *homepage*, *backend='custom'*)

**get_sorted_version_objects**()
Return list of all version objects stored, sorted from newest to oldest.

**Returns** List of version objects

**Return type** `list` of `anitya.lib.versions.Base`

**get_time_last_created_version**()
Returns creation time of latest version sorted by time of creation.

**Returns**

**Time of the latest created version or None,** if project doesn't have any version yet.

**Return type** (*arrow.Arrow*)

**get_version_class**()
Get the class for the version scheme used by this project.

This will take into account the defaults set in the ecosystem, backend, and globally. The version scheme locations are checked in the following order and the first non-null result is returned:

1. On the project itself in the `version_scheme` column.
2. The project's ecosystem default, if the project is part of one.
3. The project's backend default, if the backend defines one.
4. The global default defined in `anitya.lib.versions.GLOBAL_DEFAULT`

**Returns** A `Version` sub-class.

**Return type** anitya.lib.versions.Version

**get_version_url**()
Returns full version url, which is used by backend.

**Returns** Version url or empty string if backend is not specified

**Return type** str

**homepage**

**id**

**insecure**

**last_check**

**latest_version**

**latest_version_cursor**

**latest_version_object**

**logs**

**name**

**next_check**

**package**

**packages**

**pre_release_filter**

**regex**

**releases_only**

**classmethod search**(*session*, *pattern*, *distro=None*, *page=None*, *count=False*)
Search the projects by their name or package name

**stable_versions**
Return list of all versions that aren't flagged as pre-release.

**Returns** List of stable version objects

**Return type** list(*anitya.lib.versions.Base*)

**classmethod updated**(*session*, *status='updated'*, *name=None*, *log=None*, *page=None*, *count=False*)
Method used to retrieve projects according to their logs and how they performed at the last cron job.

**Keyword Arguments**

- **status** – used to filter the projects based on how they performed at the last cron run
- **name** – if present, will return the entries having the matching name
- **log** – if present, will return the entries having the matching log
- **page** – The page number of returned, pages contain 50 entries
- **count** – A boolean used to return either the list of entries matching the criterias or just the COUNT of entries

**updated_on**

**validate_backend**(*key*, *value*)

**version_filter**

**version_pattern**

**version_prefix**

**version_scheme**

**version_url**

**versions**
Return list of all versions stored, sorted from newest to oldest.

**Returns** List of versions

**Return type** `list` of `str`

**versions_obj**

**class** anitya.db.models.**ProjectFlag**(***kwargs*)
Bases: sqlalchemy.orm.decl_api.Base

**classmethod all**(*session*, *page=None*, *count=False*)

**created_on**

**classmethod get**(*session*, *flag_id*)

**id**

**project**

**project_id**

**reason**

**classmethod search**(*session*, *project_name=None*, *from_date=None*, *user=None*, *state=None*, *limit=None*, *offset=None*, *count=False*)
Return the list of the last Flag entries present in the database.

**Parameters**

- **cls** – the class object
- **session** – the database session used to query the information.

**Keyword Arguments**

- **project_name** – the name of the project to restrict the flags to.
- **from_date** – the date from which to give the entries.
- **user** – the name of the user to restrict the flags to.
- **state** – the flag's status (open or closed).
- **limit** – limit the result to X rows.
- **offset** – start the result at row X.
- **count** – a boolean to return the result of a COUNT query if true, returns the data if false (default).

**state**

**updated_on**

**user**

**class** anitya.db.models.**ProjectVersion**(***kwargs*)
Bases: sqlalchemy.orm.decl_api.Base

Models of version table representing version on project.

**project_id**
Related project id.

**Type** sa.Integer

**version**
Raw version string as obtained from upstream.

**Type** sa.String

**created_on**
When the version was created in Anitya.

**Type** sa.DateTime

**commit_url**
URL to commit. Currently only used by GitHub backend.

**Type** sa.String

**project**
Back reference to project.

**Type** sa.orm.relationship

**commit_url**

**created_on**

**pre_release**
Is the version pre-release?

**Returns** Pre-release flag.

**Return type** (Boolean)

**project**

**project_id**

**version**

**class** anitya.db.models.**Run**(***kwargs*)
Bases: sqlalchemy.orm.decl_api.Base

**created_on**

**error_count**

**classmethod last_entry**(*session*)
Return the last log about the cron run.

**ratelimit_count**

**success_count**

**total_count**

**class** anitya.db.models.**User**(***kwargs*)
Bases: sqlalchemy.orm.decl_api.Base

A table for Anitya users.

This table is intended to work with a table of third-party authentication providers. Anitya does not support local users.

**id**
The primary key for the table.

**Type** uuid.UUID

**email**
The user's email.

**Type** str

**username**
The user's username, as retrieved from third-party authentication.

**Type** str

**active**
Indicates whether the user is active. If false, users will not be able to log in.

**Type** bool

**admin**
Determine if this user is an administrator. If True the user is administrator.

**Type** bool

**social_auth**
The list of `social_flask_sqlalchemy.models.UserSocialAuth` entries for this user.

**Type** sqlalchemy.orm.dynamic.AppenderQuery

**active**

**admin**

**api_tokens**

**email**

**get_id**()
Implement the flask-login interface for retrieving the user's ID.

**Returns** The Unicode string that uniquely identifies a user.

**Return type** six.text_type

**id**

**is_active**
Implement the flask-login interface for determining if the user is active.

If a user is _not_ active, they are not allowed to log in.

**Returns** True if the user is active.

**Return type** bool

**is_admin**
Determine if this user is an administrator. Set admin flag if the user is preconfigured.

**Returns** True if the user is an administrator.

**Return type** bool

**is_anonymous**
Implement the flask-login interface for determining if the user is authenticated.

flask-login uses an "anonymous user" object if there is no authenticated user. This indicates to flask-login this user is not an anonymous user.

**Returns** False in all cases.

**Return type** bool

**is_authenticated**
Implement the flask-login interface for determining if the user is authenticated.

In this case, if flask-login has an instance of `User`, then that user has already authenticated via a third-party authentication mechanism.

**Returns** True in all cases.

**Return type** bool

**social_auth**

**to_dict**()
Creates json compatible dict from *User*.

**Returns** *User* object transformed to dictionary.

**Return type** dict

**username**

### Backend API

The Anitya backends API.

**class** anitya.lib.backends.**BaseBackend**
Bases: object

The base class that all the different backends should extend.

**name**
The backend name. This is displayed to the user and used in URLs.

**Type** str

**examples**
A list of strings that are displayed to the user to indicate example project URLs.

**Type** list

**default_regex**
A regular expression to use by default with the backend.

**Type** str

**more_info**
A string that provides more detailed information to the user about the backend.

**Type** str

**default_version_scheme**
The default version scheme for this backend. This is only used if both the project and the ecosystem the project is a part of do not define a default version scheme. If this is not defined, `anitya.lib.versions.GLOBAL_DEFAULT` is used.

**Type** str

**check_interval**
Interval which is used for periodic checking for new versions. This could be overriden by backend plugin.

**Type** *datetime.timedelta*

**classmethod call_url**(*url*, *last_change=None*, *insecure=False*)
Dedicated method to query a URL.

It is important to use this method as it allows to query them with a defined user-agent header thus informing the projects we are querying what our intentions are.

To prevent downloading the whole content of the page each time the url is called. We are using If-modified-since header field.

**url**
The url to request (get).
**Type** str

**last_change**
Time when the latest version was obtained. This value is used in If-modified-since header field. If there is no value provided we will use start of the epoch (1.1. 1970).
**Type** *arrow.Arrow*, optional

**insecure**
Flag for secure/insecure connection. Defaults to False.
**Type** bool, optional

**Returns** In case of FTP url it returns binary encoded string otherwise `requests.Response` object.

**classmethod check_feed**()
Method called to retrieve the latest uploads to a given backend, via, for example, RSS or an API.

Not all backends may support this. It can be used to look for updates much more quickly than scanning all known projects.

**Returns** A list of 4-tuples, containing the project name, homepage, the backend, and the version.

**Return type** `list`

**Raises**

- `AnityaPluginException` – A *anitya.lib.exceptions.AnityaPluginException* exception when the versions cannot be retrieved correctly
- `NotImplementedError` – If backend does not support batch updates.

**check_interval = datetime.timedelta(seconds=3600)**

**default_regex = None**

**default_version_scheme = None**

**examples = None**

**classmethod expand_subdirs**(*url*, *last_change=None*, *glob_char='*'*)
Expand dirs containing `glob_char` in the given URL with the latest Example URL: `https://www.example.com/foo/*/`

The globbing char can be bundled with other characters enclosed within the same slashes in the URL like `/rel*/`.

Code originally from Till Maas as part of cnucnu

**classmethod filter_versions**(*versions*, *filter_string*)

Method called to filter versions list by filter_string. Filter string is first parsed by delimiter and then applied on list of versions. For example: list of versions ["1.0.0", "1.0.0-alpha", "1.0.0-beta"] when filtered by "alpha;beta" will return ["1.0.0"].

**versions**

List of versions. For example ["1.0.0, 1.0.0-alpha"]

**Type** list

**filter_string**

String to use for filtering. It contains list of strings delimited by ";".

**Type** str

**Returns** A list of filtered versions.

**Return type** list

**classmethod get_ordered_versions**(*project*)

Method called to retrieve all the versions (that can be found) of the projects provided, ordered from the oldest to the newest.

**project**

Project object whose backend corresponds to the current plugin.

**Type** anitya.db.models.Project

**Returns** A sorted list of all the possible releases found

**Return type** list

**Raises** AnityaPluginException – A *anitya.lib.exceptions.AnityaPluginException* exception when the versions cannot be retrieved correctly

**classmethod get_version**(*project*)

Method called to retrieve the latest version of the projects provided, project that relies on the backend of this plugin.

**project**

Project object whose backend corresponds to the current plugin.

**Type** anitya.db.models.Project

**Returns** Latest version found upstream

**Return type** str

**Raises** AnityaPluginException – A *anitya.lib.exceptions.AnityaPluginException* exception when the versions cannot be retrieved correctly

**classmethod get_version_url**(*project*)

Method called to retrieve the url used to check for new version of the project provided, project that relies on the backend of this plugin.

**project**

Project object whose backend corresponds to the current plugin.

**Type** anitya.db.models.Project

**Returns** url used for version checking

**Return type** str

**classmethod get_versions**(*project*)
Method called to retrieve all the versions (that can be found) of the projects provided, project that relies on the backend of this plugin.

**project**
Project object whose backend corresponds to the current plugin.
**Type** anitya.db.models.Project

**Returns**

**A list of all the possible releases found. The items** in the list can either be strings of versions or dictionaries containing at minimum the version (in a *version* key), but also additional information like an opaque *cursor* to identify a specific version.

**Return type** list

**Raises** AnityaPluginException – A *anitya.lib.exceptions.AnityaPluginException* exception when the versions cannot be retrieved correctly

**more_info = None**

**name = None**

anitya.lib.backends.**get_versions_by_regex**(*url*, *regex*, *project*, *insecure=False*)
For the provided url, return all the version retrieved via the specified regular expression.

anitya.lib.backends.**get_versions_by_regex_for_text**(*text*, *url*, *regex*, *project*)
For the provided text, return all the version retrieved via the specified regular expression.

### Plugin API

Module handling the load/call of the plugins of anitya.

anitya.lib.plugins.**load_all_plugins**(*session*)
Load all the plugins and insert them in the database if they are not already present.

anitya.lib.plugins.**load_plugins**(*session*, *family='backends'*)
Calls load_all_plugins, but only retuns plugins specified by family argument

**Parameters** **family** (*str*) – family of the plugins, that should be returned

### Ecosystem API

The Anitya ecosystems API.

**Authors:** Nick Coghlan <ncoghlan@redhat.com>

**class** anitya.lib.ecosystems.**BaseEcosystem**
Bases: object

The base class that all the different ecosystems should extend.

**name**
The ecosystem name. This name is used to associate projects with an ecosystem and is user-facing. It is also used in URLs.

**Type** str

**default_backend**
The default backend to use for projects in this ecosystem if they don't explicitly define one to use.

**Type** str

**default_version_scheme**
The default version scheme to use for projects in this ecosystem if a they don't explicitly define one to use.

**Type** str

**aliases**
A list of alternate names for this ecosystem. These should be lowercase.

**Type** list

**aliases = []**

**default_backend = None**

**default_version_scheme = None**

**name = None**

### Versions API

The Anitya versions API.

**class** anitya.lib.versions.base.**Version**(*version: Optional[str] = None*, *prefix: Optional[str] = None*, *created_on: Optional[datetime.datetime] = None*, *pattern: Optional[str] = None*, *cursor: Optional[str] = None*, *commit_url: Optional[str] = None*, *pre_release_filter: Optional[str] = None*)

Bases: object

The base class for versions.

**name = 'Generic Version'**

**newer**(*other_versions*)
Check a version against a list of other versions to see if it's newer.

#### Example

```
>>> version = Version(version='1.1.0')
>>> version.newer([Version(version='1.0.0')])
True
>>> version.newer(['1.0.0', '0.0.1'])  # You can pass strings!
True
>>> version.newer(['1.2.0', '2.0.1'])
False
```

**Parameters** **other_versions** (*list*) – A list of version strings or Version objects to check the *version* string against.

**Returns** True if self is the newest version, False otherwise.

**Return type** bool

**Raises** InvalidVersion – if one or more of the version strings provided cannot be parsed.

**parse**()
Parse the version string to an object representing the version.

This does some minimal string processing, stripping any prefix set on project.

**Returns** The version string. Sub-classes may return a different type. object: Sub-classes may return a special class that represents the version. This must support comparison operations and return a parsed, prefix-stripped version when `__str__` is invoked.

**Return type** str

**Raises** `InvalidVersion` – If the version cannot be parsed.

**postrelease**()
Check if a version is a post-release version.

This basic version implementation does not have a concept of post-releases.

**prerelease**()
Check if a version is a pre-release version.

This basic version implementation does not have a concept of pre-releases.

anitya.lib.versions.base.**v_prefix = re.compile('v\\d.*')**
A regular expression to determine if the version string contains a 'v' prefix.

## 3.2 Development Guide

Anitya welcomes contributions! Our issue tracker is located on GitHub.

### 3.2.1 Contribution Guidelines

When you make a pull request, someone from the fedora-infra organization will review your code. Please make sure you follow the guidelines below:

#### Python Support

Anitya supports Python 3.6 or greater so please ensure the code you submit works with these versions. The test suite will run against all supported Python versions to make this easier.

#### Code Style

We follow the PEP8 style guide for Python. The test suite includes a test that enforces the required style, so all you need to do is run the tests to ensure your code follows the style. If the unit test passes, you are good to go!

To automatically format the code run the following in project root. The `.tox` folder will be created when `tox` will be run.

```
.tox/format/bin/black .
```

#### Unit Tests

The test suites can be run using tox by simply running `tox` from the repository root. These tests include unit tests, a linter to ensure Python code style is correct, checks for possible security issues, and checks the documentation for Sphinx warnings or errors.

All tests must pass. All new code should have 100% test coverage. Any bugfix should be accompanied by one or more unit tests to demonstrate the fix. If you are unsure how to write unit tests for your code, we will be happy to help you during the code review process.

#### Documentation

Anitya uses sphinx to create its documentation. New packages, modules, classes, methods, functions, and attributes all should be documented using "Google style" docstrings. For historical reasons you may encounter plain reStructuredText-style docstrings. Please consider converting them and opening a pull request!

Python API documentation is automatically generated from the code using Sphinx's autodoc extension. HTTP REST API documentation is automatically generated from the code using the httpdomain extension.

#### Release notes

To add entries to the release notes, create a file in the `news` directory with the `source.type` name format, where `type` is one of:

- `feature`: for new features
- `bug`: for bug fixes
- `api`: for API changes
- `dev`: for development-related changes
- `author`: for contributor names
- `other`: for other changes

And where the `source` part of the filename is:

- `42` when the change is described in issue `42`
- `PR42` when the change has been implemented in pull request `42`, and there is no associated issue
- `username` for contributors (`author` extention). It should be the username part of their commit's email address.

For example:

If this PR is solving bug 714 the file inside `news` should be called `714.bug` and the content of the file would be:

```
Javascript error on add project page
```

Matching the issue title.

The text inside the file will be used as entry text. A preview of the release notes can be generated with `towncrier --draft`.

### 3.2.2 Development Environment

There are two options for setting up a development environment. If you're not sure which one to choose, pick the Vagrant method.

#### Vagrant

The Vagrant development environment is set up using Ansible.

To get started, install Vagrant and Ansible. On Fedora:

```
$ sudo dnf install vagrant libvirt vagrant-libvirt vagrant-sshfs ansible
```

Next, clone the repository and start the Vagrant machine:

```
$ git clone https://github.com/fedora-infra/anitya.git
$ cd anitya
$ vagrant up
$ vagrant reload
$ vagrant ssh
```

When you log in you'll be presented with a message of the day with more details about the environment.

To start the Anitya instance in vagrant you can run:

```
$ systemctl --user start anitya
```

You may then access Anitya on your host at:

```
http://127.0.0.1:5000
```

By default, Anitya imports the production database so you've got something to start with. If instead you prefer an empty database, add the following to the Ansible provisioner inside your *Vagrantfile*:

```
ansible.extra_vars = { import_production_database: false }
```

**Note:** Please don't commit any local changes to Vagrantfile. We are managing it upstream.

Vagrant is using PostgreSQL database. To work with it use `psql` command:

```
$ sudo -u postgres psql
postgres=#\connect anitya
```

After this you can use standard SQL queries or another `psql` commands:

```
# Show description of tables
\dt
# Show table description
\d users
```

For additional `psql` commands see `man psql`.

To run libraries.io service simply run:

```
$ librariesio_consumer.py
```

To run check service simply run:

```
$ check_service.py
```

### Python virtualenv

Anitya can also be run in a Python virtualenv. For Fedora:

```
$ git clone https://github.com/fedora-infra/anitya.git
$ cd anitya
$ sudo dnf install python3-virtualenvwrapper
$ source /usr/bin/virtualenvwrapper.sh
$ mkvirtualenv anitya
$ workon anitya
```

Issuing that last command should change your prompt to indicate that you are operating in an active virtualenv.

Next, install Anitya:

```
(anitya)$ pip install -r test_requirements.txt
(anitya)$ pip install -e .
```

Create the database, by default it will be a sqlite database located at `/var/tmp/anitya-dev.sqlite`:

```
(anitya) $ python createdb.py
```

You can start the development web server included with Flask with:

```
(anitya)$ FLASK_APP=anitya.wsgi flask run
```

If you want to change the application's configuration, create a valid configuration file and start the application with the `ANITYA_WEB_CONFIG` environment variable set to the configuration file's path. You can look at the sample configuration for guidance.

### 3.2.3 Release Guide

#### Testing before release

To test the new version before release just update the `staging` branch to current `master`:

```
git checkout staging
git rebase master
git push origin/staging
```

This will automatically start the deployment in staging instance. You can then test the new changes there.

If you need to do any changes in configuration of `staging` instance, just update the release-monitoring role in Fedora infra ansible repository.

If the changes are merged, you can run the playbook by following configuration guide for Anitya in Fedora infra documentation.

---

**Note:** Have in mind that everything needs to be only done for staging. In configuration use jinja statements and when deploying don't forget to use `-l staging` switch.

---

#### Anitya

To do the release you need following python packages installed:

```
wheel
twine
towncrier
```

If you are a maintainer and wish to make a release, follow these steps:

1. Change the version in `anitya.__init__.__version__`. This is used to set the version in the documentation project and the setup.py file.
2. Add any missing news fragments to the `news` folder.
3. Get authors of commits by `python get-authors.py`.

**Note:** This script must be executed in `news` folder, because it creates files in current working directory.

4. Generate the changelog by running `towncrier`.

**Note:** If you added any news fragment in the previous step, you might see `towncrier` complaining about removing them, because they are not committed in git. Just ignore this and remove all of them manually; release notes will be generated anyway.

5. Remove every remaining news fragment from `news` folder.
6. Generate new DB schema image by running `./generate_db_schema` in `docs` folder.
7. Commit your changes with message *Anitya <version>*.
8. Tag a release with `git tag -s <version>`.
9. Don't forget to `git push --tags`.
10. Sometimes you need to also do `git push`.
11. Build the Python packages with `python setup.py sdist bdist_wheel`.
12. Upload the packages with `twine upload dist/<dists>`.

#### Fedora messaging schema

To do the release you need following python packages installed:

```
wheel
twine
```

If you are a maintainer and wish to make a release of Anitya fedora messaging schema, follow these steps:

1. Enter `anitya_schema` directory.
2. Change the version in `setup.py`.
3. Commit your changes with message *Anitya schema <version>*.
4. Don't forget to `git push`.
5. Build the Python packages with `python setup.py sdist bdist_wheel`.
6. Upload the packages with `twine upload dist/<dists>`.

## 3.3 Database models

Click below to explore Anitya's database models:

| | |
|---|---|
| *anitya.db.models* | SQLAlchemy database models. |

### 3.3.1 anitya.db.models

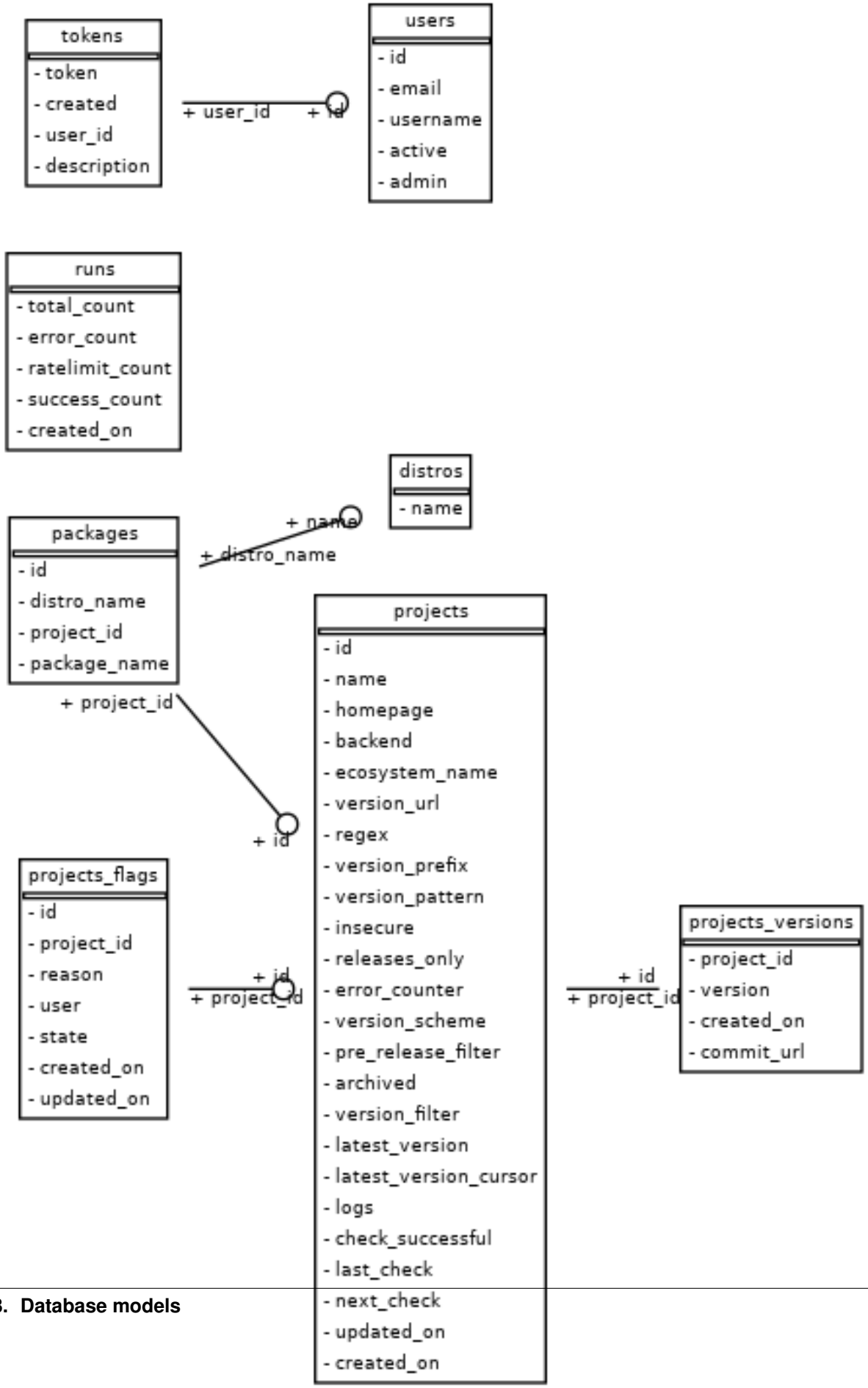
SQLAlchemy database models.

#### Classes

| | |
|---|---|
| ApiToken(**kwargs) | A table for user API tokens. |
| Distro(name) | |
| GUID(*args, **kwargs) | Platform-independent GUID type. |
| Packages(**kwargs) | |
| Project(**kwargs) | Models an upstream project and maps it to a database table. |
| ProjectFlag(**kwargs) | |
| ProjectVersion(**kwargs) | Models of version table representing version on project. |
| Run(**kwargs) | |
| User(**kwargs) | A table for Anitya users. |

### 3.3.2 Database Schema

The Anitya database schema can be seen below.


tokens
- token
- created
- user_id
- description
+ user_id
+ id
users
- id
- email
- username
- active
- admin
runs
- total_count
- error_count
- ratelimit_count
- success_count
- created_on
distros
- name
+ name
packages
- id
- distro_name
- project_id
- package_name
+ distro_name
+ project_id
+ id
projects
- id
- name
- homepage
- backend
- ecosystem_name
- version_url
- regex
- version_prefix
- version_pattern
- insecure
- releases_only
- error_counter
- version_scheme
- pre_release_filter
- archived
- version_filter
- latest_version
- latest_version_cursor
- logs
- check_successful
- last_check
- next_check
- updated_on
- created_on
projects_flags
- id
- project_id
- reason
- user
- state
- created_on
- updated_on
+ id
+ project_id
+ id
+ project_id
projects_versions
- project_id
- version
- created_on
- commit_url

# Python Module Index

## a

# HTTP Routing Table

## /api

# Index

## L



## M



## N



## P



## Q



## R



## S



## T



## U



## V