Anitya Documentation

Release 0.14.1

Red Hat, Inc. and others

Jan 17, 2019

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 ZeroMQ message via fedmsg. 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/release-monitoring/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.

1.1.1 Creating a New Project

You can create new projects in the web interface if you have 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.

Backends

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

The backends available are:

- `cpan.py` for perl projects hosted on CPAN
- `cran.py` for R projects hosted on CRAN
- `crates.py` for crates hosted on crates.io
- `debian.py` for projects hosted on the Debian ftp
- `drupal6.py` for Drupal6 modules hosted on drupal.org
- `drupal7.py` for Drupal7 modules hosted on drupal.org

- `folder.py` for projects whose release archives are provided in a folder basic apache folder or modified one.
- `freshmeat.py` for projects hosted on freshmeat.net / freecode.com
- `github.py` for projects hosted on github.com using *Github v4 API <https://developer.github.com/v4/>*
- `gitlab.py` for projects hosted on *GitLab server <https://about.gitlab.com/>_*. This backend is using *GitLab API v4 <https://docs.gitlab.com/ee/api/README.html>_*
- `gnome.py` for projects hosted on download.gnome.org
- `gnu.py` for projects hosted on gnu.org
- `google.py` for projects hosted on code.google.com
- `hackage.py` for projects hosted on hackage.haskell.org
- `launchpad.py` for projects hosted on launchpad.net
- `npmjs.py` for projects hosted on npmjs.org
- `pear.py` for projects hosted on pear.php.net
- `pecl.py` for projects hosted on pecl.php.net
- `pypi.py` for projects hosted on pypi.python.org
- `rubygems.py` for projects hosted on rubygems.org
- `sourceforge.py` for projects hosted on sourceforge.net
- `stackage.py` 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.py` for projects who require a more flexible way of finding their version.

The custom backend requires two arguments:

- `version_url` 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` 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:

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

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.

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`.

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-`.

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.

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`.

1.1.2 Integrating with Anitya

fedmsg

fedmsg 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. For the full list of messages Anitya sends, see the fedmsg topic documentation.

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

- install `fedmsg` the way you want:

On Fedora

```
dnf install fedmsg
```

On Debian

```
apt-get install fedmsg
```

Via pip

```
pip install fedmsg
```

- in the configuration file: `/etc/fedmsg.d/endpoints.py`, make sure you activate the anitya endpoint

```
"anitya-public-relay": [
    "tcp://release-monitoring.org:9940",
],
```

From python

```
import fedmsg

# Yield messages as they're available from a generator
for name, endpoint, topic, msg in fedmsg.tail_messages():
    print topic, msg
```

From the shell

```
$ fedmsg-tail --really-pretty
```

1.1.3 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 Release Notes

1.2.1 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

1.2.2 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

1.2.3 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

1.2.4 v0.13.1

Features

- Add database schema generation (#603).

Bug Fixes

- Fix cron issues (#613).

1.2.5 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).

1.2.6 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).

1.2.7 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.

1.2.8 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]

1.2.9 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)

1.3 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 of an Anitya deployment.

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.
permanent_session_lifetime = 3600
```

(continues on next page)

(continued from previous page)

```
# 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"
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.YahooOpenId",
    "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 = []

# 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
github_access_token = "foobar"

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

(continues on next page)

(continued from previous page)

```
    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"]
```

2.1.3 Services

Anitya is made up of a WSGI application, an update script that should be run at regular intervals via cron or similar, an optional fedmsg consumer, and requires a database.

WSGI Application

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

Update Script

The script that checks for project updates is located at `files/anitya_cron.py` in the git repository and Python package.

Libraries.io fedmsg Consumer

This optional service listens to a ZeroMQ socket for messages published by the libraries.io service. This requires that a bridge from Server-Sent Events (SSE) to ZeroMQ be running.

To run this, set the `anitya.libraryio.enabled` key to `True` in your fedmsg configuration and run the `fedmsg-hub` service.

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.

CHAPTER 3

Developer Guide

3.1 API Documentation

Anitya provides several APIs for users.

3.1.1 HTTP API v2

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
```

(continues on next page)

(continued from previous page)

```
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"
            "project": "requests",
            "ecosystem": "pypi",
        }
```

(continues on next page)

(continued from previous page)

```
    ],
    "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

{
    "backend": "PyPI",
    "created_on": 1490543790.0,
    "homepage": "http://python-requests.org",
```

(continues on next page)

(continued from previous page)


```
    "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.

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

(continues on next page)

(continued from previous page)

```
  ]
}
```

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": [
    {
```

(continues on next page)

(continued from previous page)


```
      "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,
      "version": "2.1.1",
      "version_url": "https://www.finnie.org/software/2ping",
      "versions": [
        "2.1.1"
      ]
    },
    {
```


(continues on next page)

(continued from previous page)


```
      "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",
        "requests-ftp",
        "requests-mock",
        "requests-ntlm",
        "requests-oauthlib",
        "requests-toolbelt"
    ],
```

(continues on next page)

(continued from previous page)

```
    "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 the latest version available from a project upstream.

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

Accepts POST queries only.

Parameters

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

Sample response:

```
{
  "backend": "Sourceforge",
  "created_on": 1409917222.0,
  "homepage": "https://sourceforge.net/projects/zero-install",
  "id": 1,
  "name": "zero-install",
  "packages": [
    {
      "distro": "Fedora",
      "package_name": "0install"
    }
  ],
  "regex": "",
  "updated_on": 1413794215.0,
  "version": "2.7",
  "version_url": "0install",
  "versions": [
    "2.7"
  ]
}
```

3.1.3 Python APIs

Anitya can be used without its web front-end, if you so choose. Please be aware that Anitya is still in the early stages of development and its Python APIs are not stable.

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
arrow.Arrow – Time when limit will be reset.

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
sqlalchemy.orm.query.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.

metadata = MetaData(bind=None)

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
object – The database objects from the query.

page
int – The page number used for the query.

items_per_page
int – The number of items per page.

total_items
int – The total number of items in the database.

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**(*session*, *flush_context*, *instances*)
An SQLAlchemy event listener that sets the ecosystem for a project if it's null.

Parameters

- **session** (*sqlalchemy.orm.session.Session*) – The session that is about to be committed.
- **flush_context** (*sqlalchemy.orm.session.UOWTransaction*) – Unused.
- **instances** (*object*) – deprecated and unused

Raises `ValueError` – If the ecosystem_name isn't valid.

anitya.db.models

SQLAlchemy database models.

class anitya.db.models.**ApiToken**(***kwargs*)
Bases: `sqlalchemy.ext.declarative.api.Base`

A table for user API tokens.

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

created
sa.DateTime – The time this API token was created.

description
sa.Text – A user-provided description of what the API token is for.

user
User – The user this API token is associated with.

created

description

token

user

user_id

class anitya.db.models.**Distro**(*name*)
Bases: sqlalchemy.ext.declarative.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.ext.declarative.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.ext.declarative.api.Base

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

id
sa.Integer – The database primary key.

name
sa.String – The upstream project's name.

homepage
sa.String – The URL for the project's home page.

backend
sa.String – The name of the backend to use when fetching updates; this is a foreign key to a Backend.

ecosystem_name
sa.String – 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.

version_url
sa.String – 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).

regex
sa.String – A Python re style regular expression that is applied to the HTML from version_url to find versions.

insecure
sa.Boolean – Whether or not to validate the x509 certificate offered by the server at version_url. Defaults to False.

latest_version
sa.Boolean – The latest version for the project, as determined by the version sorting algorithm.

logs
sa.Text – The result of the last update.

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

updated_on
sa.DateTime – When the project was last updated.

created_on
sa.DateTime – When the project was created in Anitya.

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

version_scheme
sa.String – 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.

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

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)*) – 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

flags

classmethod get(*session*, *project_id*)

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.db.models.ProjectVersion*

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

logs

name

next_check

package

packages

regex

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

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_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.ext.declarative.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.ext.declarative.api.Base

created_on

project

project_id

version

class anitya.db.models.**Run**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

created_on

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

status

class anitya.db.models.**User**(***kwargs*)
Bases: sqlalchemy.ext.declarative.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
uuid.UUID – The primary key for the table.

email
str – The user's email.

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

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

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

social_auth
sqlalchemy.orm.dynamic.AppenderQuery – The list of social_flask_sqlalchemy.models.UserSocialAuth entries for this user.

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

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

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

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

default_version_scheme
str – 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.

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

classmethod call_url(*url*, *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.

url
str – The url to request (get).

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

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(0, 3600)

default_regex = None

default_version_scheme = None

examples = None

classmethod expand_subdirs(*url*, *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 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
anitya.db.models.Project – Project object whose backend corresponds to the current plugin.

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
anitya.db.models.Project – Project object whose backend corresponds to the current plugin.

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
anitya.db.models.Project – Project object whose backend corresponds to the current plugin.

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
anitya.db.models.Project – Project object whose backend corresponds to the current plugin.

Returns A 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

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

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

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

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

aliases = []

default_backend = None

default_version_scheme = None

name = None

Versions API

The Anitya versions API.

class anitya.lib.versions.base.**Version**(*version=None*, *prefix=None*, *created_on=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 release-monitoring organization will review your code. Please make sure you follow the guidelines below:

Python Support

Anitya supports Python 2.7 and Python 3.4 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!

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, 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.

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 configure your Vagrantfile:

```
$ git clone https://github.com/release-monitoring/anitya.git
$ cd anitya
$ cp Vagrantfile.example Vagrantfile
$ vagrant up
$ vagrant reload
$ vagrant ssh
```

You may then access Anitya on your host at:

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

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

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

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

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
anitya=\#\dt
# Show table description
anitya=\#\d users
```

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

To run the cron job in Vagrant guest run these commands:

```
$ workon anitya
$ python files/anitya_cron.py
```

Python virtualenv

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

```
$ git clone https://github.com/release-monitoring/anitya.git
$ cd anitya
$ sudo dnf install python3-virtualenvwrapper
$ 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-env)$ pip install -r test_requirements.txt
(anitya-env)$ pip install -e .
```

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

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

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

```
(anitya-env)$ 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.

Listening for local event announcements

To listen for local event announcements over the Federated Message Bus, first start a local relay in the background:

```
$ fedmsg-relay --config-filename fedmsg.d/fedmsg-config.py &
```

And then display the received messages in the local console:

```
$ fedmsg-tail --config fedmsg.d/fedmsg-config.py --no-validate --really-pretty
```

These commands will pick up the local config automatically if you're in the Anitya checkout directory, but being explicit ensures they don't silently default to using the global configuration.

To display the messages, we turn off signature validation (since the local server will be emitting unsigned messages) and pretty-print the received JSON.

Refer to the fedmsg consumer API for more details on receiving event messages programmatically.

Tips

Anitya publishes fedmsgs, and these are viewable with `fedmsg-tail`:

```
$ workon anitya
$ fedmsg-tail
```

This will also show you all incoming messages from libraries.io's SSE feed.

3.2.3 Release Guide

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.
5. Commit your changes.
6. Tag a release with `git tag -s <version>`.
7. Don't forget to `git push --tags`.
8. Build the Python packages with `python setup.py sdist bdist_wheel`.
9. 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)	
Run(**kwargs)	
User(**kwargs)	A table for Anitya users.

3.3.2 Database Schema

The Anitya database schema can be seen below.

Fig. 1: Database schema.

Python Module Index

a

HTTP Routing Table

/api

Index

Q

R

S

T

U

V