Pulp Python Support Documentation

Release 2.0a1

Pulp Project

Apr 11, 2019

Contents

The `pulp_python` plugin extends pulpcore to support hosting pip compatible Python packages. This plugin is a part of the Pulp Project, and assumes some familiarity with the pulpcore documentation.

If you are just getting started, we recommend getting to know the *basic workflows*.

Community contributions are encouraged.

- Send us pull requests on our GitHub repository.
- View and file issues in the Redmine Tracker.

CHAPTER 1

REST API

REST API documentation for this plugin can be found here

CHAPTER 2

Table of Contents

2.1 User Setup

2.1.1 Install pulp-python

This document assumes that you have installed pulpcore into a the virtual environment `pulpvenv`.

Users should install from **either** PyPI or source.

From PyPI

```
sudo -u pulp -i
source ~/pulpvenv/bin/activate
pip install pulp-python
```

From Source

```
sudo -u pulp -i
source ~/pulpvenv/bin/activate
git clone https://github.com/pulp/pulp_python.git
cd pulp_python
pip install -e .
```

2.1.2 Make and Run Migrations

```
pulp-manager makemigrations python
pulp-manager migrate python
```

2.1.3 Run Services

```
pulp-manager runserver
gunicorn pulpcore.content:server --bind 'localhost:8080' --worker-class 'aiohttp.
↪GunicornWebWorker' -w 2
sudo systemctl restart pulp-resource-manager
sudo systemctl restart pulp-worker@1
sudo systemctl restart pulp-worker@2
```

2.2 Workflows

If you have not yet installed the Python plugins on your Pulp installation, please follow our *User Setup*. These documents will assume you have the environment installed and ready to go.

The REST API examples here use httpie to perform the requests. The `httpie` commands below assume that the user executing the commands has a `.netrc` file in the home directory. The `.netrc` should have the following configuration:

```
machine localhost
login admin
password admin
```

If you configured the `admin` user with a different password, adjust the configuration accordingly. If you prefer to specify the username and password with each request, please see `httpie` documentation on how to do that.

To make these workflows copy/pastable, we make use of environment variables. The first variable to set is the hostname and port:

```
$ export BASE_ADDR=http://<hostname>:8000
```

This documentation makes use of the jq library to parse the json received from requests, in order to get the unique urls generated when objects are created. To follow this documentation as-is please install the jq library with:

```
$ sudo dnf install jq
```

2.2.1 Synchronize a Repository

Users can populate their repositories with content from an external source like PyPI by syncing their repository.

Create a Repository

Start by creating a new repository named “foo”:

```
$ http POST $BASE_ADDR/pulp/api/v3/repositories/ name=foo
```

Response:

```
{
    "_href": "/pulp/api/v3/repositories/1/",
    ...
}
```

If you want to copy/paste your way through the guide, create an environment variable for the repository URI:

```
$ export REPO_HREF=$(http $BASE_ADDR/pulp/api/v3/repositories/ | jq -r '.results[] |␣
↪select(.name == "foo") | ._href')
```

Create a Remote

Creating a remote object informs Pulp about an external content source. In this case, we will be using a fixture, but Python remotes can be anything that implements the PyPI API. This can be PyPI itself, a fixture, or even an instance of Pulp 2.

You can use any Python remote to sync content into any repository:

```
$ http POST $BASE_ADDR/pulp/api/v3/remotes/python/python/ \
    name='bar' \
    url='https://pypi.org/' \
    includes:='[{"name": "django", "version_specifier":"~=2.0"}]'
```

Response:

```
{
    "_href": "/pulp/api/v3/repositories/foo/remotes/python/python/1/",
    ...
}
```

Again, you can create an environment variable for convenience:

```
$ export REMOTE_HREF=$(http $BASE_ADDR/pulp/api/v3/remotes/python/python/ | jq -r '.
↪results[] | select(.name == "bar") | ._href')
```

A More Complex Remote

If only the name of a project is specified, every distribution of every version of that project will be synced. You can use the version_specifier and digest fields on a project to ensure only distributions you care about will be synced:

```
$ http POST $BASE_ADDR/pulp/api/v3/remotes/python/python/ \
    name='complex-remote' \
    url='https://pypi.org/' \
    includes:='[
        { "name": "django",
          "version_specifier": "~=2.0,!=2.0.1",
          "digests":[
                {"type": "sha256",
                 "digest":
↪"3d9916515599f757043c690ae2b5ea28666afa09779636351da505396cbb2f19"}
          ]
        },
        {"name": "pip-tools",
         "version_specifier": ">=1.12,<=2.0"},
        {"name": "scipy",
         "digests":[
            {"type": "md5",
            "digest": "044af71389ac2ad3d3ece24d0baf4c07"},
            {"type": "sha256",
            "digest":
↪"18b572502ce0b17e3b4bfe50dcaea414a98290358a2fa080c36066ba0651ec14"}]
```

(continues on next page)

(continued from previous page)

```
        },
        {"name": "shelf-reader"}
    ]'
```

You can also use version specifiers to “exclude” certain versions of a project, like so:

```
$ http POST $BASE_ADDR/pulp/api/v3/remotes/python/python/ \
    name='complex-remote' \
    url='https://pypi.org/' \
    includes:='[
        {"name": "django", "version_specifier": ""},
        {"name": "scipy", "version_specifier": ""}
    ]' \
    excludes:='[
        {"name": "django", "version_specifier": "~=1.0"},
        {"name": "scipy", "digests":[
            {"type": "md5",
            "digest": "044af71389ac2ad3d3ece24d0baf4c07"},
            {"type": "sha256",
            "digest":
↪"18b572502ce0b17e3b4bfe50dcaea414a98290358a2fa080c36066ba0651ec14"}]
        },
    ]'
```

Sync repository foo with remote

Use the remote object to kick off a synchronize task by specifying the repository to sync with. You are telling pulp to fetch content from the remote and add to the repository.

By default Pulp syncs using `mirror` sync. This *adds* new content from the remote repository and *removes* content from the local repository until the local repository “mirrors” the remote. You can also tell Pulp not to mirror, and Pulp will only *add* new content from the remote repository to the local repository:

```
$ http POST $BASE_ADDR$REMOTE_HREF'sync/' repository=$REPO_HREF mirror=False
```

Response:

```
{
    "task": "/pulp/api/v3/tasks/3896447a-2799-4818-a3e5-df8552aeb903/"
}
```

You can follow the progress of the task with a GET request to the task href. Notice that when the synchroinze task completes, it creates a new version, which is specified in `created_resources`:

```
$  http $BASE_ADDR/pulp/api/v3/tasks/3896447a-2799-4818-a3e5-df8552aeb903/
```

Response:

```
{
    "_href": "/pulp/api/v3/tasks/3896447a-2799-4818-a3e5-df8552aeb903/",
    "created": "2018-05-01T17:17:46.558997Z",
    "created_resources": [
        "/pulp/api/v3/repositories/1/versions/6/"
    ],
    "error": null,
```

(continues on next page)

(continued from previous page)

```
    "finished_at": "2018-05-01T17:17:47.149123Z",
    "non_fatal_errors": [],
    "parent": null,
    "progress_reports": [
        {
            "done": 0,
            "message": "Add Content",
            "state": "completed",
            "suffix": "",
            "task": "/pulp/api/v3/tasks/3896447a-2799-4818-a3e5-df8552aeb903/",
            "total": 0
        },
        {
            "done": 0,
            "message": "Remove Content",
            "state": "completed",
            "suffix": "",
            "task": "/pulp/api/v3/tasks/3896447a-2799-4818-a3e5-df8552aeb903/",
            "total": 0
        }
    ],
    "spawned_tasks": [],
    "started_at": "2018-05-01T17:17:46.644801Z",
    "state": "completed",
    "worker": "/pulp/api/v3/workers/eaffe1be-111a-421d-a127-0b8fa7077cf7/"
}
```

2.2.2 Upload and Manage Content

Create a repository

If you don't already have a repository, create one:

```
$ http POST $BASE_ADDR/pulp/api/v3/repositories/ name=foo
```

Response:

```
{
    "_href": "/pulp/api/v3/repositories/1/",
    ...
}
```

Create a variable for convenience:

```
$ export REPO_HREF=$(http $BASE_ADDR/pulp/api/v3/repositories/ | jq -r '.results[] |␣
↪select(.name == "foo") | ._href')
```

Upload a file to Pulp

Each artifact in Pulp represents a file. They can be created during sync or created manually by uploading a file:

```
$ export ARTIFACT_HREF=$(http --form POST $BASE_ADDR/pulp/api/v3/artifacts/ file@./
↪shelf_reader-0.1-py2-none-any.whl | jq -r '._href')
```

Response:

```
{
    "_href": "/pulp/api/v3/artifacts/1/",
    ...
}
```

Create content from an artifact

Now that Pulp has the wheel, its time to make it into a unit of content. The python plugin will inspect the file and populate its metadata:

```
$ http POST $BASE_ADDR/pulp/api/v3/content/python/packages/ _artifact=$ARTIFACT_HREF␣
↪filename=shelf_reader-0.1-py2-none-any.whl
```

Response:

```
{
    "_href": "/pulp/api/v3/content/python/packages/1/",
    "_artifact": "/pulp/api/v3/artifacts/1/",
    "digest": "b5bb9d8014a0f9b1d61e21e796d78dccdf1352f23cd32812f4850b878ae4944c",
    "filename": "shelf_reader-0.1-py2-none-any.whl",
    "type": "python"
}
```

Create a variable for convenience:

```
$ export CONTENT_HREF=$(http $BASE_ADDR/pulp/api/v3/content/python/packages/ | jq -r
↪'.results[] | select(.filename == "shelf_reader-0.1-py2-none-any.whl") | ._href')
```

Add content to a repository

Once there is a content unit, it can be added and removed and from to repositories:

```
$ http POST $BASE_ADDR$REPO_HREF'versions/' add_content_units:="[\"$CONTENT_HREF\"]"
```

2.2.3 Publish and Host

This section assumes that you have a repository with content in it. To do this, see the *Synchronize a Repository* or *Upload and Manage Content* documentation.

Create a Publisher

Publishers contain extra settings for how to publish. You can use a Python publisher on any repository that contains Python content:

```
$ http POST $BASE_ADDR/pulp/api/v3/publishers/python/python/ name=bar
```

Response:

```
{
    "_href": "/pulp/api/v3/repositories/foo/publishers/python/python/1/",
    ...
}
```

Create a variable for convenience.:

```
$ export PUBLISHER_HREF=$(http $BASE_ADDR/pulp/api/v3/publishers/python/python/ | jq -
↪r '.results[] | select(.name == "bar") | ._href')
```

Publish a repository with a publisher

Use the remote object to kick off a publish task by specifying the repository version to publish. Alternatively, you can specify repository, which will publish the latest version.

The result of a publish is a publication, which contains all the information needed for `pip` to use. Publications are not consumable until they are hosted by a distribution:

```
$ http POST $BASE_ADDR$PUBLISHER_HREF'publish/' repository=$REPO_HREF
```

Response:

```
{
    "task": "/pulp/api/v3/tasks/fd4cbecd-6c6a-4197-9cbe-4e45b0516309/"
}
```

Create a variable for convenience.:

```
$ export PUBLICATION_HREF=$(http $BASE_ADDR/pulp/api/v3/publications/ | jq -r --arg␣
↪PUBLISHER_HREF "$PUBLISHER_HREF" '.results[] | select(.publisher==$PUBLISHER_HREF)␣
↪| ._href')
```

Host a Publication (Create a Distribution)

To host a publication, (which makes it consumable by `pip`), users create a distribution which will serve the associated publication at `/pulp/content/<distribution.base_path>` as demonstrated in *using distributions*:

```
$ http POST $BASE_ADDR/pulp/api/v3/distributions/ name='baz' base_path='foo'␣
↪publication=$PUBLICATION_HREF
```

Response:

```
{
    "_href": "/pulp/api/v3/distributions/1/",
   ...
}
```

Use the newly created distribution

The metadata and packages can now be retrieved from the distribution:

```
$ http $BASE_ADDR/pulp/content/foo/simple/
$ http $BASE_ADDR/pulp/content/foo/simple/shelf-reader/
```

The content is also pip installable:

```
$ pip install --trusted-host localhost -i $BASE_ADDR/pulp/content/foo/simple/ shelf-
↪reader
```

If you don't want to specify the distribution path every time, you can modify your `pip.conf` file. See the pip docs for more detail.:

```
$ cat pip.conf
```

```
[global]
index-url = http://localhost:8080/pulp/content/foo/simple/
```

The above configuration informs `pip` to install from `pulp`:

```
$ pip install --trusted-host localhost shelf-reader
```

2.3 pulp-python 3.0 Release Notes

pulp-python 3.0 is currently in Beta. Backwards incompatible changes might be made until Beta is over.

2.3.1 3.0.0b4

- Adds support for pulpcore 3.0.0.rc1.
- Adds excludes support (aka 'blacklist')

  Renames the "projects" field on the remote to "includes".

  Adds a new "excludes" field to the remote which behaves like "includes", except that any specified releasees or digests are not synced, even if an include specifier matches them.

  Also adds a 'prereleases' field to the remote, which toggles whether prerelease versions should be synced. This mirrors the 'prereleases' flag that packaging.specifiers.SpecifierSet provides.
- Removes Python 3.5 support

CHAPTER 3

Indices and tables

- genindex
- modindex
- search