Devmason Server Documentation

Release 1.0

Eric Holscher & Jacob Kaplan-Moss

Nov 02, 2018

Contents

This is a server that is meant to be used for reporting test results of tests. Currently it's main focus is on Python, but there's no reason that it can't support other types of test results.

Contents:

CHAPTER 1

Build server REST API

This is a proposed standard for a REST API for build clients to use to communicate with a build server. It's inspired by pony-build, and generally rather Python-oriented, but the goal is language-agnostic.

Contents

1.1 API Usage

1.1.1 Registering a new project

```
-> PUT /{project}

   {Project}

<- 201 Created
   Location: /{project}/builds/{build-id}
```

If a project already exists, a 403 Forbidden will be returned.

Users may register with authentication via HTTP Basic:

```
-> PUT /{project}
   Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==

   {Project_}

<- 201 Created
   Location: /{project}/builds/{build-id}
```

If this is done, then that authorization may be repeated in the future to update/delete the project or to delete builds. No explicit user registration step is needed; users will be created on the fly.

Warning: Since the authorization uses HTTP Basic, build servers should probably support SSL for the security-conscious.

1.1.2 Reporting a build

```
-> POST /{project}/builds

   {Build}

<- 201 Created
   Location: /{project}/builds/{build-id}
```

1.1.3 Incremental build reporting

```
-> POST /{project}/builds

   {Incremental build}
```

```
<- 201 Created
   Location: /{project}/builds/{build-id}/progress

-> POST /{project}/builds/{build-id}/progress

   {Build step}

<- 204 No Content
   Location: /{project}/builds/{build-id}

-> POST /{project}/builds/{build-id}/progress

   {Build step}

<- 204 No Content
   Location: /{project}/builds/{build-id}


...

-> DELETE /{project}/builds/{build-id}/progress

<- 204 No Content
   Location: /{project}/builds/{build-id}
```

1.2 API Reference

1.2.1 Representation formats

- JSON.
- UTF-8.
- All datetimes in RFC 2822.

1.2.2 URIs

URI	Resource	Methods	Notes
`/`	*Project list*	GET	
`/{project}`	*Project*	`GET`, `PUT`, `DELETE`	Only the user that created a project may update (`PUT`) or delete it.
`/{project}/builds`	*Build list*	`GET`, `POST`	
`/{project}/builds/latest`	–	`GET`	302 redirect to latest build.
`/{project}/builds/{build-id}`	*Build*	`GET`, `PUT`, `DELETE`	Builds may not be updated; `PUT` only exists if clients wish for some reason to use a predetermined build id. Only the user that created a build or the project owner may delete a build.
`/{project}/builds/{build-id}/progress`	*Build progress*	`GET`, `POST`, `DELETE`	
`/{project}/tags`	*Tag list*	`GET`	
`/{project}/tags/{-listjoin|-|tags}`	*Build list*	`GET`	
`/{project}/tags/{-listjoin|-|tags}/latest`	–	`GET`	302 redirect to latest build given tags
`/users`	*User list*	`GET`	
`/users/{username}`	*User*	`GET`, `PUT`, `DELETE`	Authentication required to `PUT` or `DELETE`.
`/users/{username}/builds`	*Build list*	`GET`	
`/users/{username}/builds/latest`	–	`GET`	302 redirect to latest build by user

All resources support `OPTIONS` which will return a list of allowed methods in the `Allow` header. This is particularly useful to check authentication for methods that require it.

1.2.3 Resources

Build

Representation:

```
{
  'success': true,                              # did the build succeed?
  'started': 'Tue, 20 Oct 2009 10:20:00 -0500',
  'finished': 'Tue, 20 Oct 2009 10:22:00 -0500,

  'tags': ['list', 'of', 'tags'],

  'client': {
    'host': 'example.com',                      # host that ran the build
    'user': 'http://example.com/'               # user to credit for build.
    'arch': 'macosx-10.5-i386'                  # architecture the build was␣
↪done on.
    ... [1]
  },

  'results': [{Build step}, ...],

  'links': [{Link}, ...]
}
```

Notes:

Links:

Rel	Links to
`self`	This *build*
`project`	The *project* this is a builds of.
`tag`	A *tag* this build is tagged with. There'll probably be many `tag` links.

Build list

Representation:

```
{
  'builds': [{Build}, ...],

  'count': 100,                   # total number of builds available
  'num_pages': 4,                 # total number of pages
  'page': 1                       # current page number
  'paginated': true               # is this list paginated?
  'per_page': 25,                 # number of builds per page

  'links': [{Link, ...}]
}
```

Links:

[1] Clients may include arbitrary extra client info in the client record.

Rel	Links to
`self`	This *build list*
`project`	The *project* this is a list of builds for (if applicable).
`user`	The *user* this is a list of builds for (if applicable).
`tag`	The *tag* this is a list of builds for (if applicable).
`latest-build`	URI for the redirect to this project's latest build.
`next`	The next page of builds (if applicable).
`previous`	The previous page of builds (if applicable).
`first`	The first page of builds.
`last`	The last page of builds.

Build progress

Used as an entry point for *incremental build reporting*

Empty representation – the existence of the resource indicates an in-progress build. When the build is done, the resource will return 410 Gone.

Build step

Representation:

```
{
  'success': true,                               # did this step succeed?
  'started': 'Tue, 20 Oct 2009 10:20:00 -0500',
  'finished': 'Tue, 20 Oct 2009 10:22:00 -0500,
  'name': 'checkout',                            # human-readable name for the␣
↪step
  'output': '...'                                # stdout for this step
  'errout': '...'                                # stderr for this step
  ... [2]
}
```

Notes:

Incremental build

`POST` this resource to a *build list* to signal the start of an incremental build.

Representation

```
{
  'incremental': true,                           # never false
  'started': 'Tue, 20 Oct 2009 10:20:00 -0500', # when the build started on
                                                 # the client (not when the
                                                 # packet was posted!)

  'client': {
    'host': 'example.com',                       # host that ran the build
    'user': 'username'                           # user to credit for build.
    'arch': 'macosx-10.5-i386'                   # architecture the build was␣
↪done on.
```

[2] Build steps may include arbitrary extra build info in the record.

```
        ... [3]
    },

    'tags': ['list', 'of', 'tags'],
}
```

Notes:

Link

Used all over the damn place to knit resources together.

Representation:

```
{
    'rel': 'self',                      # identifier for the type of link this is
    'href': 'http://example.com/',      # full URL href
    'allowed_methods': ['GET'],         # list of methods this client can perform on said␣
↪resource
}
```

Project

Representation:

```
{
    'name': 'Project Name',
    'owner': 'username',         # the user who created the project, if applicable.

    'links': [{Link}, ...]
}
```

Links:

Rel	Links to
`self`	This *project*.
`build-list`	This project's *build list*.
`latest-build`	URI for the redirect to this project's latest build.
`tag-list`	This project's *tag list*.

Project list

```
{
    'projects': [{Project}, ...],
    'links': [{Link}, ...]
}
```

Links:

Rel	Links to
`self`	This server.

[3] Clients may include arbitrary extra client info in the client record.

Tag

Tag detail.

```
{
  'tags': ['list', 'of', 'tags'],        # Or just a single ['tag'] if this
                                          # is one tag.

  'builds': [{Build}, ...],

  'count': 100,                           # total number of builds w/this tag
  'num_pages': 4,                         # total number of pages
  'page': 1                               # current page number
  'paginated': true                       # is this list paginated?
  'per_page': 25,                         # number of builds per page

  'links': [{Link, ...}]
}
```

Links:

Tag list

Representation:

```
{
  'tags': ['tag1', 'tag2', 'tag3'],
  'links': [{Link, ...}]
}
```

Links:

Rel	Links to
`self`	This *tag list*
`project`	The *project* in question.
`tag`	Each *tag* used by the project gets a link.

User

Representation:

```
{
  'username': 'username',
  'links': [{Link}, ...]
}
```

Links:

Rel	Links to
`self`	This *user*
`builds`	*Build list* for this user.

User list

Representation:

```
{
  'users': [{User}, ...],

  'count': 100,                  # total number of users available
  'num_pages': 4,                # total number of pages
  'page': 1                      # current page number
  'paginated': true              # is this list paginated?
  'per_page': 25,                # number of users per page
  'links': [{Link, ...}]
}
```

Links:

Rel	Links to
`self`	This *user*

CHAPTER 2

How to install Devmason Server

Devmason is easy to install. It comes with a setup.py, so you can easily install it:

```
virtualenv devmason
cd devmason/
. bin/activate
pip install -e git://github.com/ericholscher/devmason-server.git#egg=devmason-server
cd src/devmason-server
pip install -r pip_requirements.txt
cd test_project
./manage.py syncdb --noinput
./manage.py loaddata devmason_server_test_data.json
./manage.py runserver
```

That's all that it takes to get a running server up. Look at the test_project for examples on how to set up your urls and settings.

CHAPTER 3

Usage

Using the server is pretty simple. Most of the interaction is done through the API, which has a basic client library. The client library is located on github: http://github.com/ericholscher/devmason-utils/.

3.1 Using the test runner

Once you have *devmason_utils* installed, it ships with it's own test runner that reports your test results to the server. Simply add the following in your settings:

```
TEST_RUNNER = 'devmason_utils.test_runner.run_tests'
PB_USER = 'your_user'
PB_PASS = 'your_pass'
```

When you do this the username will be created for your on the server, then your results should automatically be sent to http://devmason.com.

Note: A username and password is only required to create a project. If you're just sending results to someone else's project then you only need to set up your test runner.