JenkinsAPI Documentation

Release 0.3.4

xxx

Mar 30, 2017

Contents

Jenkins is the market leading continuous integration system, originally created by Kohsuke Kawaguchi. This API makes Jenkins even easier to use by providing an easy to use conventional python interface.

Jenkins (and It's predecessor Hudson) are fantastic projects - but they are somewhat Java-centric. Thankfully the designers have provided an excellent and complete REST interface. This library wraps up that interface as more conventional python objects in order to make most Jenkins oriented tasks simpler.

This library can help you:

- Query the test-results of a completed build
- Get a objects representing the latest builds of a job
- Search for artifacts by simple criteria
- Block until jobs are complete
- Install artifacts to custom-specified directory structures
- Username/password auth support for jenkins instances with auth turned on
- Search for builds by subversion revision
- Add, remove and query jenkins slaves

CHAPTER 1

Sections

User API

This module is a collection of helpful, high-level functions for automating common tasks. Many of these functions were designed to be exposed to the command-line, hence they have simple string arguments.

jenkinsapi.api.**block_until_complete**(*jenkinsurl*, *jobs*, *maxwait=12000*, *interval=30*, *raise_on_timeout=True*, *username=None*, *password=None*, *ssl_verify=True*)
Wait until all of the jobs in the list are complete.

jenkinsapi.api.**get_artifacts**(*jenkinsurl*, *jobid=None*, *build_no=None*, *username=None*, *password=None*, *ssl_verify=True*)
Find all the artifacts for the latest build of a job.

jenkinsapi.api.**get_build**(*jenkinsurl*, *jobname*, *build_no*, *username=None*, *password=None*, *ssl_verify=True*)
A convenience function to fetch down the test results from a jenkins job by build number.

jenkinsapi.api.**get_latest_build**(*jenkinsurl*, *jobname*, *username=None*, *password=None*, *ssl_verify=True*)
A convenience function to fetch down the very latest test results from a jenkins job.

jenkinsapi.api.**get_latest_complete_build**(*jenkinsurl*, *jobname*, *username=None*, *password=None*, *ssl_verify=True*)
A convenience function to fetch down the very latest test results from a jenkins job.

jenkinsapi.api.**get_latest_test_results**(*jenkinsurl*, *jobname*, *username=None*, *password=None*, *ssl_verify=True*)
A convenience function to fetch down the very latest test results from a jenkins job.

jenkinsapi.api.**get_nested_view_from_url**(*url*, *username=None*, *password=None*, *ssl_verify=True*)
Returns View based on provided URL. Convenient for nested views.

jenkinsapi.api.**get_view_from_url**(*url*, *username=None*, *password=None*, *ssl_verify=True*)
Factory method

jenkinsapi.api.**grab_artifact**(*jenkinsurl*, *jobid*, *artifactid*, *targetdir*, *username=None*, *password=None*, *strict_validation=False*, *ssl_verify=True*)
Convenience method to find the latest good version of an artifact and save it to a target directory. Directory is made automatically if not exists.

jenkinsapi.api.**install_artifacts**(*artifacts*, *dirstruct*, *installdir*, *basestaticurl*, *strict_validation=False*)
Install the artifacts.

jenkinsapi.api.**search_artifact_by_regexp**(*jenkinsurl*, *jobid*, *artifactRegExp*, *username=None*, *password=None*, *ssl_verify=True*)
Search the entire history of a hudson job for a build which has an artifact whose name matches a supplied regular expression. Return only that artifact.

@param jenkinsurl: The base URL of the jenkins server @param jobid: The name of the job we are to search through @param artifactRegExp: A compiled regular expression object

(not a re-string)

@param username: Jenkins login user name, optional @param password: Jenkins login password, optional

jenkinsapi.api.**search_artifacts**(*jenkinsurl*, *jobid*, *artifact_ids=None*, *username=None*, *password=None*, *ssl_verify=True*)
Search the entire history of a jenkins job for a list of artifact names. If same_build is true then ensure that all artifacts come from the same build of the job

Artifact

Artifacts can be used to represent data created as a side-effect of running a Jenkins build.

Artifacts are files which are associated with a single build. A build can have any number of artifacts associated with it.

This module provides a class called Artifact which allows you to download objects from the server and also access them as a stream.

class jenkinsapi.artifact.**Artifact**(*filename*, *url*, *build*, *relative_path=None*)
Represents a single Jenkins artifact, usually some kind of file generated as a by-product of executing a Jenkins build.

get_data()
Grab the text of the artifact

save(*fspath*, *strict_validation=False*)
Save the artifact to an explicit path. The containing directory must exist. Returns a reference to the file which has just been writen to.

Parameters **fspath** – full pathname including the filename, str

Returns filepath

save_to_dir(*dirpath*, *strict_validation=False*)
Save the artifact to a folder. The containing directory must exist, but use the artifact's default filename.

Build

A jenkins build represents a single execution of a Jenkins Job.

Builds can be thought of as the second level of the jenkins heirarchy beneath Jobs. Builds can have state, such as whether they are running or not. They can also have outcomes, such as wether they passed or failed.

Build objects can be associated with Results and Artifacts.g

class jenkinsapi.build.**Build**(*url*, *buildno*, *job*, *depth=1*)
Represents a jenkins build, executed in context of a job.

get_causes()
Returns a list of causes. There can be multiple causes lists and some of the can be empty. For instance, when a build is manually aborted, Jenkins could add an empty causes list to the actions dict. Empty ones are ignored.

get_changeset_items()
Returns a list of changeSet items.

Each item has structure as in following example: {

"affectedPaths": ["content/rcm/v00-rcm-xccdf.xml"

], "author" : {

"absoluteUrl": "http://jenkins_url/user/username79", "fullName": "username"

}, "commitId": "3097", "timestamp": 1414398423091, "date": "2014-10-27T08:27:03.091288Z", "msg": "commit message", "paths": [{

"editType": "edit", "file": "/some/path/of/changed_file"

}], "revision": 3097, "user": "username"

}

get_console()
Return the current state of the text console.

get_downstream_builds()
Get the downstream builds for this build :return List of Build or None

get_downstream_job_names()
Get the downstream job names for this build :return List of string or None

get_downstream_jobs()
Get the downstream jobs for this build :return List of jobs or None

get_env_vars()
Return the environment variables.

This method is using the Environment Injector plugin: https://wiki.jenkins-ci.org/display/JENKINS/EnvInject+Plugin

get_master_build()
Get the master build if it exist, None otherwise :return Build or None

get_master_build_number()
Get the master build number if it exist, None otherwise :return: int or None

get_master_job()
Get the master job object if it exist, None otherwise :return: Job or None

get_master_job_name()
Get the master job name if it exist, None otherwise :return: String or None

get_matrix_runs()
For a matrix job, get the individual builds for each matrix configuration :return: Generator of Build

get_params()
Return a dictionary of params names and their values or None if no parameters present

get_result_url()
Return the URL for the object which provides the job's result summary.

get_resultset()
Obtain detailed results for this build.

get_timestamp()
Returns build timestamp in UTC

get_upstream_build()
Get the upstream build if it exist, None otherwise :return Build or None

get_upstream_build_number()
Get the upstream build number if it exist, None otherwise :return: int or None

get_upstream_job()
Get the upstream job object if it exist, None otherwise :return: Job or None

get_upstream_job_name()
Get the upstream job name if it exist, None otherwise :return: String or None

has_resultset()
Return a boolean, true if a result set is available. false if not.

is_good()
Return a bool, true if the build was good. If the build is still running, return False.

is_running()
Return a bool if running.

stop()
Stops the build execution if it's running :return boolean True if succeded False otherwise or the build

is not running

Using Jenkins API

JenkinsAPI lets you query the state of a running Jenkins server. It also allows you to change configuration and automate minor tasks on nodes and jobs.

Example 1: Get version of Jenkins

```
from jenkinsapi.jenkins import Jenkins

def get_server_instance():
    jenkins_url = 'http://jenkins_host:8080'
    server = Jenkins(jenkins_url, username='foouser', password='foopassword')
    return server

if __name__ == '__main__':
    print get_server_instance().version
```

The above code prints version of Jenkins running on the host *jenkins_host*.

From Jenkins vesion 1.426 onward one can specify an API token instead of your real password while authenticating the user against Jenkins instance. Refer to the the Jenkis wiki page Authenticating scripted clients for details about how a user can generate an API token. Once you have API token you can pass the API token instead of real password while creating an Jenkins server instance using Jenkins API.

Example 2: Get details of jobs running on Jenkins server

```
"""Get job details of each job that is running on the Jenkins instance"""
def get_job_details():
    # Refer Example #1 for definition of function 'get_server_instance'
    server = get_server_instance()
    for job_name, job_instance in server.get_jobs():
        print 'Job Name:%s' % (job_instance.name)
        print 'Job Description:%s' % (job_instance.get_description())
        print 'Is Job running:%s' % (job_instance.is_running())
        print 'Is Job enabled:%s' % (job_instance.is_enabled())
```

Example 3: Disable/Enable a Jenkins Job

```
"""Disable a Jenkins job"""
def disable_job():
    # Refer Example #1 for definition of function 'get_server_instance'
    server = get_server_instance()
    job_name = 'nightly-build-job'
    if (server.has_job(job_name)):
        job_instance = server.get_job(job_name)
        job_instance.disable()
        print 'Name:%s,Is Job Enabled ?:%s' % (job_name,job_instance.is_enabled())
```

Use the call `job_instance.enable()` to enable a Jenkins Job.

Example 4: Get Plugin details

Below chunk of code gets the details of the plugins currently installed in the Jenkins instance.

```
def get_plugin_details():
    # Refer Example #1 for definition of function 'get_server_instance'
    server = get_server_instance()
    for plugin in server.get_plugins().values():
        print "Short Name:%s" % (plugin.shortName)
        print "Long Name:%s" % (plugin.longName)
        print "Version:%s" % (plugin.version)
        print "URL:%s" % (plugin.url)
        print "Active:%s" % (plugin.active)
        print "Enabled:%s" % (plugin.enabled)
```

Example 5: Getting version information from a completed build

This is a typical use of JenkinsAPI - it was the very first use I had in mind when the project was first built: In a continuous-integration environment you want to be able to programatically detect the version-control information of the last succsessful build in order to trigger some kind of release process.:

```
from jenkinsapi.jenkins import Jenkins

def getSCMInfroFromLatestGoodBuild(url, jobName, username=None, password=None):
    J = Jenkins(url, username, password)
    job = J[jobName]
    lgb = job.get_last_good_build()
```

```
    return lgb.get_revision()

if __name__ == '__main__':
    print getSCMInfroFromLatestGoodBuild('http://localhost:8080', 'fooJob')
```

When used with the Git source-control system line 20 will print out something like '8b4f4e6f6d0af609bb77f95d8fb82ff1ee2bba0d' - which looks suspiciously like a Git revision number.

Rules for Contributors

The JenkinsAPI project welcomes contributions via GitHub. Please bear in mind the following guidelines when preparing your pull-request.

Python compatibility

The project currently targets Python 2.6 and Python 2.7. Support for Python 3.x will be introduced soon. Please do not add any features which will break our supported Python 2.x versions or make it harder for us to migrate to Python 3.x

Test Driven Development

Please do not submit pull requests without tests. That's really important. Our project is all about test-driven development. It would be embarrasing if our project failed because of a lack of tests!

You might want to follow a typical test driven development cycle: http://en.wikipedia.org/wiki/Test-driven_development

Put simply: Write your tests first and only implement features required to make your tests pass. Do not let your implementation get ahead of your tests.

Features implemented without tests will be removed. Unmaintained features (which break because of changes in Jenkins) will also be removed.

Check the CI status before comitting

We have a Travis CI account - please verify that your branch works before making a pull request.

Any problems?

If you are stuck on something, please post to the issue tracker. Do not contact the developers directly.

Important Links

Support & bug-reportst https://github.com/salimfadhley/jenkinsapi/issues?direction=desc&sort=comments&state=open

Project source code github: https://github.com/salimfadhley/jenkinsapi

Project documentation https://jenkinsapi.readthedocs.org/en/latest/

Releases http://pypi.python.org/pypi/jenkinsapi

Installation

Egg-files for this project are hosted on PyPi. Most Python users should be able to use pip or setuptools to automatically install this project.

Most users can do the following:

```
pip install jenkinsapi
```

Or..

```
   easy_install jenkinsapi

* In Jenkins > 1.518 you will need to disable "Prevent Cross Site Request Forgery␣
↪exploits".
* Remember to set the Jenkins Location in general settings - Jenkins' REST web-
↪interface will not work if this is set incorrectly.
```

Examples

JenkinsAPI is intended to map the objects in Jenkins (e.g. Builds, Views, Jobs) into easily managed Python objects:

```
Python 2.7.4 (default, Apr 19 2013, 18:28:01)
[GCC 4.7.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import jenkinsapi
>>> from jenkinsapi.jenkins import Jenkins
>>> J = Jenkins('http://localhost:8080')
>>> J.keys() # Jenkins objects appear to be dict-like, mapping keys (job-names) to
['foo', 'test_jenkinsapi']
>>> J['test_jenkinsapi']
<jenkinsapi.job.Job test_jenkinsapi>
>>> J['test_jenkinsapi'].get_last_good_build()
<jenkinsapi.build.Build test_jenkinsapi #77>
```

JenkinsAPI lets you query the state of a running Jenkins server. It also allows you to change configuration and automate minor tasks on nodes and jobs.

You can use Jenkins to get information about recently completed builds. For example, you can get the revision number of the last succsessful build in order to trigger some kind of release process.:

```
from jenkinsapi.jenkins import Jenkins

def getSCMInfroFromLatestGoodBuild(url, jobName, username=None, password=None):
    J = Jenkins(url, username, password)
    job = J[jobName]
    lgb = job.get_last_good_build()
    return lgb.get_revision()

if __name__ == '__main__':
    print getSCMInfroFromLatestGoodBuild('http://localhost:8080', 'fooJob')
```

When used with the Git source-control system line 20 will print out something like '8b4f4e6f6d0af609bb77f95d8fb82ff1ee2bba0d' - which looks suspiciously like a Git revision number.

Note: As of Jenkins version 1.426, and above, an API token can be specified instead of your real password, while authenticating the user against the Jenkins instance. Refer to the the Jenkis wiki page [Authenticating scripted clients](https://wiki.jenkins-ci.org/display/JENKINS/Authenticating+scripted+clients) for details about how a user can generate an API token. Once you have obtained an API token you can pass the API token instead of real password while creating an Jenkins server instance using Jenkins API.

Tips & Tricks

Getting the installed version of JenkinsAPI

This package supports PEP-396 by implementing a __version__ attribute. This contains a string in the format x.y.z:

```
>>> import jenkinsapi
>>> print(jenkinsapi.__version__)
0.2.23
```

There is also a command-line tool for use in the shell:

```
$ jenkinsapi_version
0.2.23
```

CHAPTER 2

Project Authors

- Salim Fadhley (sal@stodge.org)
- Ramon van Alteren (ramon@vanalteren.nl)
- Ruslan Lutsenko (ruslan.lutcenko@gmail.com)

Plus many others, please see the README file for a more complete list of contributors and how to contact them.

CHAPTER 3

Extending and Improving JenkinsAPI

JenkinsAPI is a pure-python project and can be improved with almost any programmer's text-editor or IDE. I'd recomend the following project layout which has been shown to work with both SublimeText2 and Eclipse/PyDev

- Make sure that pip and virtualenv are installed on your computer. On most Linux systems these can be installed directly by the OS package-manager.
- Create a new virtualenv for the project:

```
virtualenv jenkinsapi
```

- Change to the new directory and check out the project code into the **src** subdirectory:

```
cd jenkinsapi
git clone https://github.com/salimfadhley/jenkinsapi.git src
```

- Activate your jenkinsapi virtual environment:

```
cd bin
source activate
```

- Install the jenkinsapi project in 'developer mode' - this step will automatically download all of the project's dependancies:

```
cd ../src
python setup.py develop
```

- Test the project - this step will automatically download and install the project's test-only dependancies. Having these installed will be helpful during development:

```
python setup.py test
```

- Set up your IDE/Editor configuration - the **misc** folder contains configuration for Sublime Text 2. I hope in time that other developers will contribute useful configurations for their favourite development tools.

Testing

The project maintainers welcome any code-contributions. Please conside the following when you contribute code back to the project:

- All contrubutions should come as github pull-requests. Please do not send code-snippets in email or as attachments to issues.
- Please take a moment to clearly describe the intended goal of your pull-request.
- Please ensure that any new feature is covered by a unit-test

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

j

Index