

Working with Python Jenkins

Javier Salas

What is Python Jenkins?

Python Jenkins is a **python wrapper** for the [Jenkins](#) REST API which aims to provide a more **conventionally pythonic way** of controlling a Jenkins server. It provides a higher-level API containing a number of convenience functions.

In terms of abstraction, we will see that instead of dealing with **json objects and http requests**, we will deal with **python objects** to interact with Jenkins

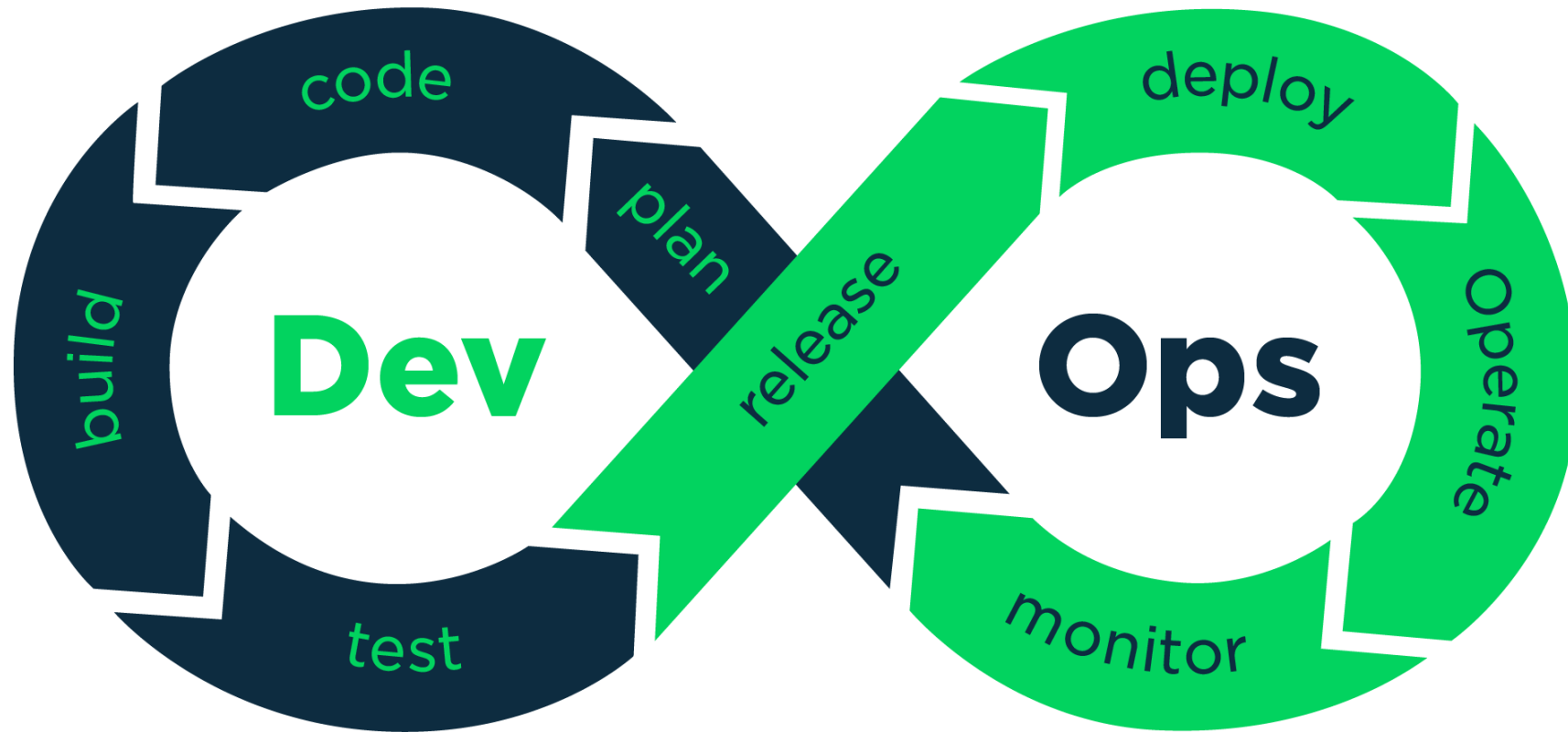


Jenkins

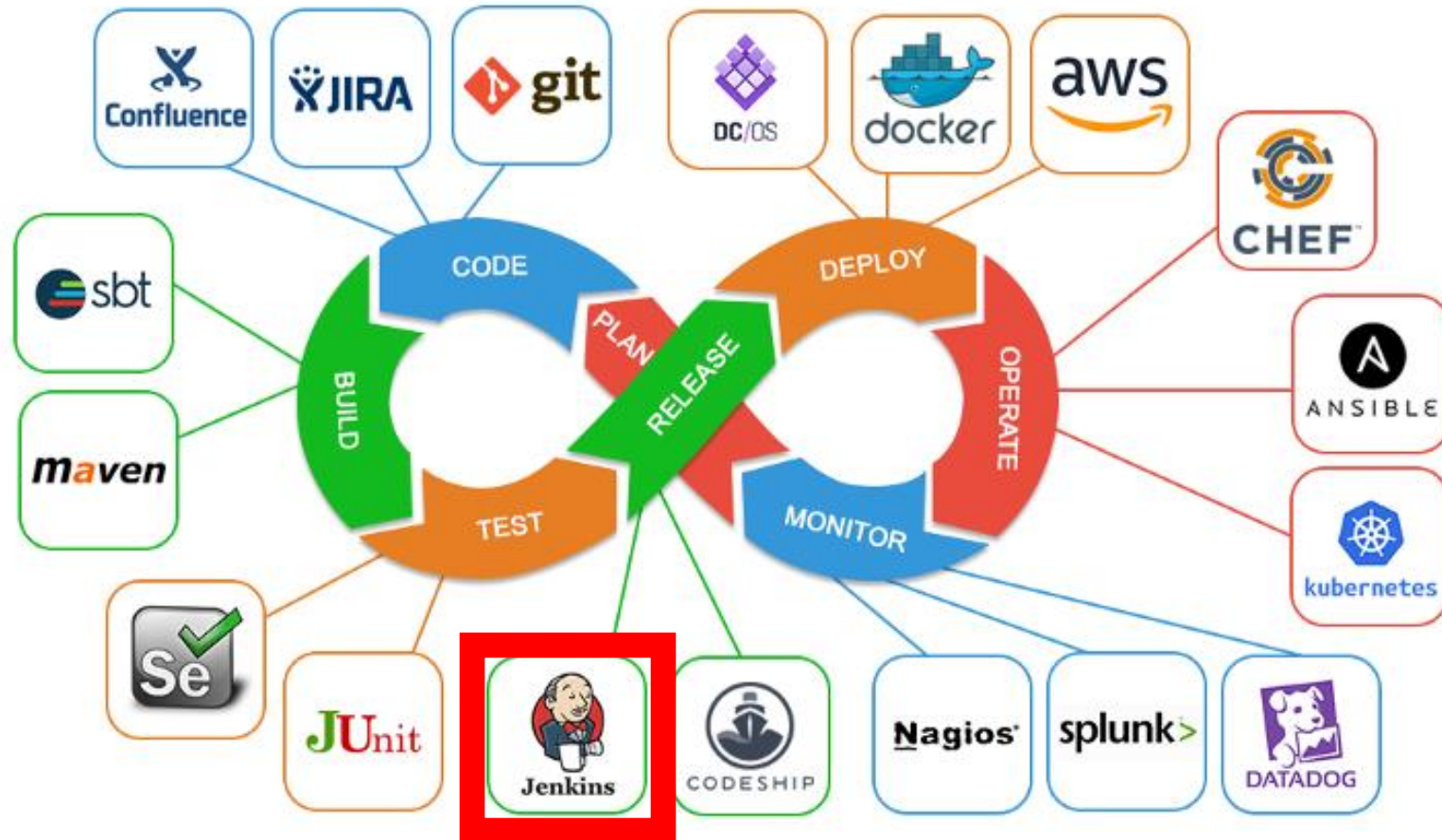
What is Jenkins?

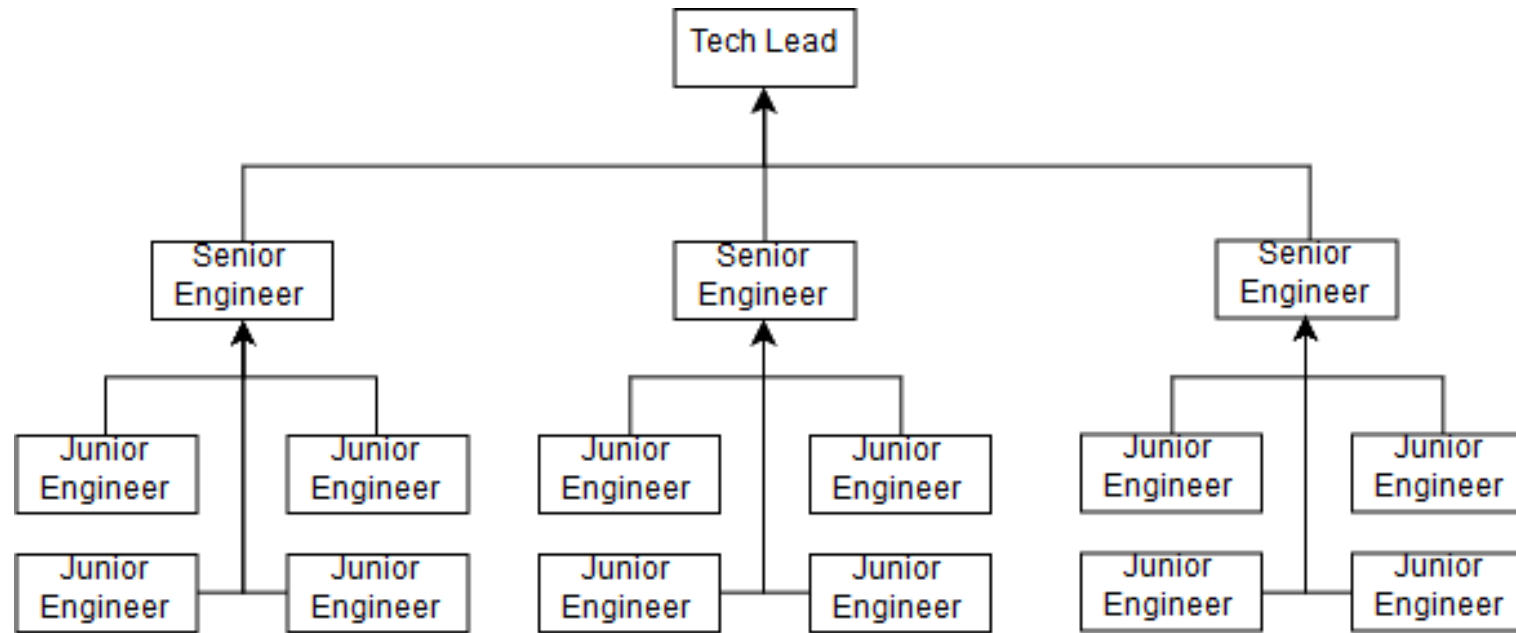
Jenkins is a *devops* tool that performs tests, listens for commits, generates reports and much more.

Big Picture: Devops



Where we are in the process:





Consider the following
scenario

- What if you are the Tech Lead at a company and you wanted make a program to improve your organization's programming.

What should the program do?

- Build stability
- Project progression
- Unit tests
- Automated reporting of errors
- Report generating periodically



Motivating factors: CI/CD

Continuous Integration:

- Test your application with every change
- Detect failures as they arise

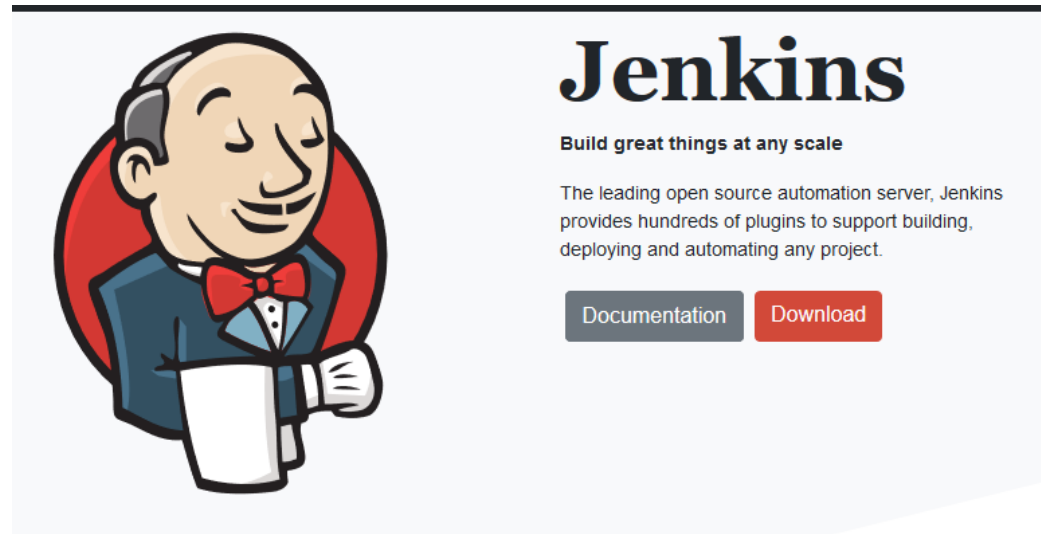
Continuous Deployment

- Automate deployments
- Limit risk of human error
- `git commit -m "a commit message"`
 - This should do everything

Getting started: Installing Jenkins and Jenkins-Python

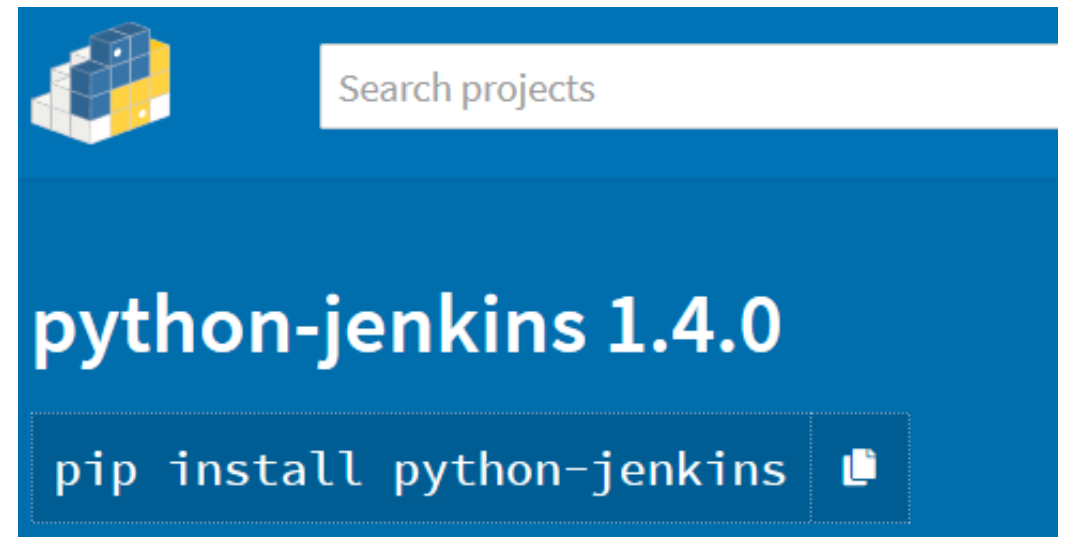
Installing Jenkins

- Download the standalone .war



Installing python-jenkins

- `pip install python-jenkins`






Starting Jenkins

- Jenkins is a Java application (.war)
 - `java -jar Jenkins.war`
- Jenkins will begin a service that will let you connect to it via a web browser.
 - In the real world, the UI doesn't need to be used since it's irrelevant to Jenkins's job
 - Along side a REST API, we also have a Jenkins client that can connect to the server which can send CLI commands.
 - Default is localhost:8080
- Out of the box, Jenkins is useless, we must configure it before we can do anything.

Jenkins Terminology

Name	Definition
Job/Project	Refers to a runnable task that are controller / monitored by Jenkins
Build	Result of one run of a Project
Node/Slave	Slaves are computers that are set up to build projects for a master. Jenkins runs a separate program called “slave agent” on slaves. The term node is used to refer to all machines that are part of the Jenkins grid
Stable build	A build is stable if it was build successfully and no publisher reports it as unstable
Unstable build	A build is unstable if it was build successfully and one or more publisher report it unstable. A test publisher may report a project as unstable
Publisher	A publisher is part of the build process other than compilation for example, Junit test runs

Identifying components of the UI

 **Jenkins**

2

admin | log out

Jenkins

[New Item](#)

[People](#)

[Build History](#)

[Manage Jenkins](#)

[My Views](#)

[Lockable Resources](#)

[Credentials](#)

[New View](#)

Build Queue

No builds in the queue.

Build Executor Status

1 Idle

2 Idle

=====MOTD=====















You're beautiful :^)

Jobs/Projects

[add description](#)

All

+


S	W	Name ↓	Last Success	Last Failure	Last Duration	
		DevProject1	1 day 17 hr - #2	1 day 17 hr - #1	38 ms	
		empty	N/A	N/A	N/A	
		empty_copy	N/A	N/A	N/A	
		JenkinsTestJob	N/A	N/A	N/A	
		TestProject1	N/A	N/A	N/A	


Icon: [S](#) [M](#) [L](#)


Build Success

Build Stability

[Legend](#)

 [RSS for all](#)

 [RSS for failures](#)

 [RSS for just latest builds](#)

Understanding a project



The screenshot shows the Jenkins web interface for a project named "DevProject1". The top navigation bar includes the Jenkins logo and the project name. A left sidebar contains links to various project actions: "Back to Dashboard", "Status", "Changes", "Workspace", "Build Now", "Delete Project", "Configure", and "Rename". The main content area is titled "Project DevProject1" and includes links to "Workspace" and "Recent Changes". Below this is a "Build History" section with a search bar and a table of builds. The table shows two builds: build #2 (successful) and build #1 (failed), both dated Feb 20, 2019. At the bottom of the build history are links for "RSS for all" and "RSS for failures". A "Permalinks" section on the right lists several links to specific build pages, such as "Last build (#2), 21 days ago".

Jenkins

Jenkins ▾ DevProject1 ▸

Back to Dashboard

Status

Changes

Workspace

Build Now

Delete Project

Configure

Rename

Project DevProject1

Workspace

Recent Changes

Build History trend

find x

#2	Feb 20, 2019 1:49 AM
#1	Feb 20, 2019 1:47 AM


Build #

RSS for all RSS for failures

Permalinks

- [Last build \(#2\), 21 days ago](#)
- [Last stable build \(#2\), 21 days ago](#)
- [Last successful build \(#2\), 21 days ago](#)
- [Last failed build \(#1\), 21 days ago](#)
- [Last unsuccessful build \(#1\), 21 days ago](#)
- [Last completed build \(#2\), 21 days ago](#)

Configuring a project

 **Jenkins**

Jenkins ▶ DevProject1 ▶

General

Source Code ManagementBuild TriggersBuild EnvironmentBuildPost-build Actions

Description

[Plain text] [Preview](#)

Looking at build

Build



Execute shell

Command `# Execute a command and see if it returns successful`
`ls -l`



Console Output

Started by user [admin](#)
Building in workspace /home/salas/.jenkins/workspace/DevProject1
[DevProject1] \$ /bin/sh -xe /tmp/jenkins5623244251682341223.sh
+ ls -l
total 0
Finished: SUCCESS

Constructor: establishing connection

```
class jenkins.Jenkins(url, username=None, password=None, timeout=<object object>, resolve=True)
```

Create handle to Jenkins instance.

All methods will raise `JenkinsException` on failure.

Parameters:

- `url` – URL of Jenkins server, `str`
- `username` – Server username, `str`
- `password` – Server password, `str`
- `timeout` – Server connection timeout in secs (default: not set), `int`
- `resolve` – Attempts to resolve and auto-correct API redirection. default: True
`bool`

Important methods: Getting all jobs

```
get_all_jobs(folder_depth=None)
```

Get list of all jobs recursively to the given folder depth.

Each job is a dictionary with 'name', 'url', 'color' and 'fullname' keys.

Parameters: `folder_depth` - Number of levels to search, `int`. By default `None`, which will search all levels. 0 limits to toplevel.

Returns: list of jobs, `[{ str: str }]`

Recommended: pout

- pip3 install pout

```
In [4]: 1 print(server.get_all_jobs())
```

```
[{'_class': 'hudson.model.FreeStyleProject', 'name': 'DevProject1', 'url': 'http://localhost:8080/job/DevProject1/', 'color': 'blue', 'fullname': 'DevProject1'}, {'_class': 'hudson.model.FreeStyleProject', 'name': 'empty', 'url': 'http://localhost:8080/job/empty/', 'color': 'disabled', 'fullname': 'empty'}, {'_class': 'hudson.model.FreeStyleProject', 'name': 'empty_copy', 'url': 'http://localhost:8080/job/empty_copy/', 'color': 'notbuilt', 'fullname': 'empty_copy'}, {'_class': 'hudson.model.FreeStyleProject', 'name': 'GitProject', 'url': 'http://localhost:8080/job/GitProject/', 'color': 'notbuilt', 'fullname': 'GitProject'}, {'_class': 'hudson.model.FreeStyleProject', 'name': 'JenkinsTestJob', 'url': 'http://localhost:8080/job/JenkinsTestJob/', 'color': 'notbuilt', 'fullname': 'JenkinsTestJob'}, {'_class': 'hudson.model.FreeStyleProject', 'name': 'TestProject1', 'url': 'http://localhost:8080/job/TestProject1/', 'color': 'notbuilt', 'fullname': 'TestProject1'}]
```

In [3]:

```
1 import pout
2 pout.v(server.get_all_jobs())
```

```
server.get_all_jobs() (6) =
[
  0:
    {
      '_class': "hudson.model.FreeStyleProject",
      'name': "DevProject1",
      'url': "http://localhost:8080/job/DevProject1/",
      'color': "blue",
      'fullname': "DevProject1"
    },
  1:
    {
      '_class': "hudson.model.FreeStyleProject",
      'name': "empty",
      'url': "http://localhost:8080/job/empty/",
      'color': "disabled",
      'fullname': "empty"
    },
  2:
    {
      '_class': "hudson.model.FreeStyleProject",
      'name': "empty_copy",
      'url': "http://localhost:8080/job/empty_copy/",
      'color': "notbuilt",
      'fullname': "empty_copy"
    },
  3:
    {
      '_class': "hudson.model.FreeStyleProject",
      'name': "GitProject",
      'url': "http://localhost:8080/job/GitProject/",
      'color': "notbuilt",
      'fullname': "GitProject"
    },
  4:
    {
      '_class': "hudson.model.FreeStyleProject",
      'name': "JenkinsTestJob",
      'url': "http://localhost:8080/job/JenkinsTestJob/",
      'color': "notbuilt",
      'fullname': "JenkinsTestJob"
    },
  5:
    {
```

Recommended:
pout

Important methods: Getting job info

```
get_job_info(name, depth=0, fetch_all_builds=False) 🔗
```

Get job information dictionary.

Parameters:

- **name** – Job name, `str`
- **depth** – JSON depth, `int`
- **fetch_all_builds** – If true, all builds will be retrieved from Jenkins. Otherwise, Jenkins will only return the most recent 100 builds. This comes at the expense of an additional API call which may return significant amounts of data. `bool`

Returns:

dictionary of job information

Example: getting job info

```
In [8]: 1 pout.v(server.get_job_info('DevProject1'))
```

```
server.get_job_info('DevProject1') (31) =  
{  
    '_class': "hudson.model.FreeStyleProject",  
    'actions':  
        [  
            0: {},  
            1: {},  
            2: {},  
            3:  
                {  
                    '_class': "com.cloudbees.plugins.credentials.ViewCredentialsAction"  
                }  
        ],  
    'description': "",  
    'displayName': "DevProject1",  
    'displayNameOrNull': None,  
    'fullDisplayName': "DevProject1",  
    'fullName': "DevProject1",  
    'name': "DevProject1"
```

Important methods: Getting build info

```
get_build_info(name, number, depth=0)
```

Get build information dictionary.

Parameters:

- `name` - Job name, `str`
- `number` - Build number, `int`
- `depth` - JSON depth, `int`

Returns:

dictionary of build information, `dict`

Example: getting build info

```
In [17]: 1 pout.v(server.get_build_info('DevProject1',3))

server.get_build_info('DevProject1',3) (20) =
{
    'artifacts': [],
    'building': False,
    'description': None,
    'displayName': "#3",
    'duration': 344,
    'estimatedDuration': 362,
    'executor': None,
    'fullDisplayName': "DevProject1 #3",
    'id': "3",
    'keepLog': False,
    'number': 3,
    'queueId': 5,
    'result': "SUCCESS",
    'timestamp': 1552492693541,
    'url': "http://localhost:8080/job/DevProject1/3/",
    'buildOn': ""
```

Important methods:

```
In [15]: 1 print(server.get_job_config('DevProject1'))

<?xml version='1.1' encoding='UTF-8'?>
<project>
  <actions/>
  <description></description>
  <keepDependencies>false</keepDependencies>
  <properties/>
  <scm class="hudson.scm.NullSCM"/>
  <canRoam>true</canRoam>
  <disabled>false</disabled>
  <blockBuildWhenDownstreamBuilding>false</blockBuildWhenDownstreamBuilding>
  <blockBuildWhenUpstreamBuilding>false</blockBuildWhenUpstreamBuilding>
  <triggers/>
  <concurrentBuild>false</concurrentBuild>
  <builders>
    <hudson.tasks.Shell>
      <command># Execute a command and see if it returns successful
ls -l</command>
    </hudson.tasks.Shell>
  </builders>
  <publishers/>
  <buildWrappers/>
</project>
```


Jenkins and Python

- Using Jenkins, we can run **nosetests**. By configuring Jenkins to convert the output of nosetests into to an xml file we can process it as a **Post-Build Action**.
- We can also use **pylint** for checking our style.



And



Build

Execute shell

Command

```
nosetests --with-xunit --all-modules --traverse-namespace --with-coverage --cover-package=project1 --cover-inclusive ;  
python -m coverage xml --include=project1* ;  
pylint -f parseable -d I0011,R0801 project1 | tee pylint.out
```

See [the list of available environment variables](#)

Add build step ▾

Building: using nosetests and
python

After installing nosetests and pylint on the machine. Install the Jenkins Cobertura Plugin (for nosetests) and Jenkins Violations (for pylint)

Post-build Actions

Publish JUnit test result report

X

?

Test report XMLs

[Fileset 'includes'](#) setting that specifies the generated raw XML report files, such as 'myproject/target/test-reports/*.xml'. Basedir of the fileset is [the workspace root](#).

☐ Retain long standard output/error

?

Health report amplification factor

1% failing tests scores as 99% health. 5% failing tests scores as 95% health

?

Allow empty results

☐ Do not fail the build on empty test results

?

Post-build action: nosetests

When nosetests runs the code coverage, it generates a .coverage file. Jenkins can't read that. The second line of the script (python -m coverage xml...) converts the .coverage file to an xml format that Jenkins Cobertura plugin can read.

Publish Cobertura Coverage Report

X

Cobertura xml report pattern

coverage.xml

This is a file name pattern that can be used to locate the cobertura xml report files (for example with Maven2 use ****/target/site/cobertura/coverage.xml**). The path is relative to the module root unless you have configured your SCM with multiple modules, in which case it is relative to the workspace root. Note that the module root is SCM-specific, and may not be the same as the workspace root.

Cobertura must be configured to generate XML reports for this plugin to function.

NOTE: If concurrent builds are enabled for this job, and a later build finishes before an earlier build, the later build will reduce or skip trend analysis/charting.

Enable New API

☐

Advanced...

Post-build action: pylint

The last line (pylint...) runs pylint on the project and outputs it in a format that the Violations Jenkins plugin can read.

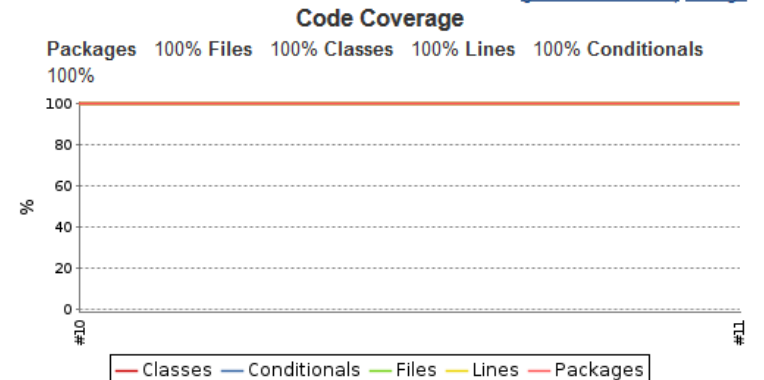
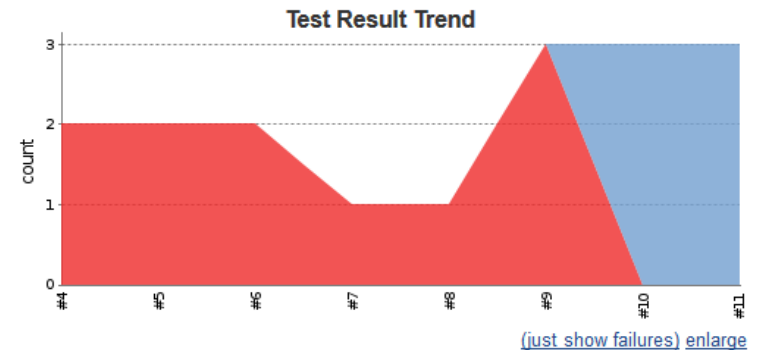
Looking at the results of our new build

Project PythonProject

[add description](#)[Disable Project](#)[Coverage Report](#)[Workspace](#)[Recent Changes](#)[Latest Test Result](#) (no failures)

Permalinks

- [Last build \(#11\), 1 hr 37 min ago](#)
- [Last stable build \(#11\), 1 hr 37 min ago](#) ▼
- [Last successful build \(#11\), 1 hr 37 min ago](#)
- [Last failed build \(#9\), 2 hr 10 min ago](#)
- [Last unsuccessful build \(#9\), 2 hr 10 min ago](#)
- [Last completed build \(#11\), 1 hr 37 min ago](#)



Important methods: Getting build test reports

```
get_build_test_report(name, number, depth=0)
```

Get test results report.

Parameters:

- name – Job name, `str`
- number – Build number, `int`

Returns:

dictionary of test report results, `dict` or None if there is no Test Report

Example: getting test reports

```
In [29]: 1 pout.v(server.get_build_test_report('PythonProject',11))
```

```
server.get_build_test_report('PythonProject',11) (8) =  
{  
  '_class': "hudson.tasks.junit.TestResult",  
  'testActions': [],  
  'duration': 0.002,  
  'empty': False,  
  'failCount': 0,  
  'passCount': 3,  
  'skipCount': 0,  
  'suites':  
    [  
      0:  
        {  
          'cases':  
            [  
              0:  
                {  
                  'testActions': [],
```

Putting it all together:

- Creating a new Jenkins project from existing source code
 - executing 'python script.py' – This script will do the following:
 - Installs Jenkins Plugins
 - Set the git path to cwd by replacing the git element using xpath
 - Starts a build
 - Prints the build's debug info


```
12 def installPlugins(plugins):
13     '''
14     Installs the plugins given in the list
15     Returns bool if restart is required
16     '''
17     plugins_list=[key[0] for key in (server.get_plugins()).keys()]
18     restart = True
19     for plugin in plugins:
20         if plugin not in plugins_list:
21             restart = restart and server.install_plugin(plugin)
22     return restart
```

Installing Jenkins Plugins

```
23 file_path='PythonProject.xml'
24 output_path='config_new.xml'
25
26 def createNewConfig(file_path,output_path):
27     '''
28     Makes a copy of the example PythonProject's build script and replaces the Git URL to the current working directory
29     '''
30     doc = etree.parse(file_path)
31     root=doc.getroot()
32
33     # The path for the node that contains the location of our git folder
34     code = root.xpath('//scm/userRemoteConfigs/hudson.plugins.git.UserRemoteConfig/url')
35
36     if code:
37         # Replaces <url> text
38         code[0].text = os.getcwd()
39         # Save back to the XML file
40         etree.ElementTree(root).write(output_path, pretty_print=True)
41
```

Replaces the git element using xpath

```
73     server.build_job(project_name)
74
75     print(server.debug_job_info(project_name))
```

Start a build and print debug info

JavierTSalas / JenkinsRepo

Unwatch

1

Star

0

Fork

0

<> Code

Issues 0

Pull requests 0

Projects 0

Wiki

Insights

Settings

Sample repo for CIS4930 for using Jenkins with nosetests and pylint

Edit

Manage topics

6 commits

1 branch

0 releases

1 contributor

Branch: master

New pull request

Create new file

Upload files

Find File

Clone or download

JavierTSalas Update README.md

Latest commit 276b4ee just now

JenkinsRepo	Renamed folders and added a check for plugins installed	7 minutes ago
tests	Renamed folders and added a check for plugins installed	7 minutes ago
PythonProject.xml	Updated xml to match renaming of folders	5 minutes ago
README.md	Update README.md	just now
script.py	Renamed folders and added a check for plugins installed	7 minutes ago

Try it for yourself

```
(py3) salas@DESKTOP-P3FFU98:/mnt/c/Users/Javier/Desktop/Python/JenkinsQuickStart/JenkinsRepo$ python scri
Hello admin from Jenkins 2.150.2
Starting JenkinsQuickStart
class hudson.model.FreeStyleProject
actions [{'_class': 'hudson.plugins.violations.ViolationsProjectAction'}, {}, {}, {}, {'_class': 'com.cl
description
displayName NewPythonProject
displayNameOrNull None
fullDisplayName NewPythonProject
fullName NewPythonProject
name NewPythonProject
url http://localhost:8080/job/NewPythonProject/
buildable True
builds []
color notbuilt
firstBuild None
healthReport []
inQueue True
keepDependencies False
lastBuild None
lastCompletedBuild None
lastFailedBuild None
lastStableBuild None
lastSuccessfulBuild None
lastUnstableBuild None
lastUnsuccessfulBuild None
nextBuildNumber 1
property []
queueItem {'_class': 'hudson.model.Queue$WaitingItem', 'blocked': False, 'buildable': False, 'id': 46, 'i
Project', 'url': 'http://localhost:8080/job/NewPythonProject/', 'url': 'queue/item/46/', 'why': 'In the
concurrentBuild False
downstreamProjects []
labelExpression None
scm {'_class': 'hudson.plugins.git.GitSCM'}
upstreamProjects []
None
```

Try it for yourself
