Using Python-Jenkins

The python-jenkins library allows management of a Jenkins server through the Jenkins REST endpoints. Below are examples to get you started using the library. If you need further help take a look at the API reference docs for more details.

Example 1: Get version of Jenkins

This is an example showing how to connect to a Jenkins instance and retrieve the Jenkins server version.

```
import jenkins

server = jenkins.Jenkins('http://localhost:8080', username='myuser', password='mypassword')
user = server.get_whoami()
version = server.get_version()
print('Hello %s from Jenkins %s' % (user['fullName'], version))
```

The above code prints the fullName attribute of the user and the version of the Jenkins master running on 'localhost:8080'. For example, it may print "Hello John from Jenkins 2.0".

From Jenkins version 1.426 onward you can specify an API token instead of your real password while authenticating the user against the Jenkins instance. Refer to the Jenkins Authentication wiki for details about how you can generate an API token. Once you have an API token you can pass the API token instead of a real password while creating a Jenkins instance.

Example 2: Logging into Jenkins using kerberos

Kerberos support is only enabled if you have "kerberos" python package installed. You can install the "kerberos" package from PyPI using the obvious pip command.

```
pip install kerberos
```

Note

This might require python header files as well as kerberos header files.

If you have “kerberos” python package installed, python-jenkins tries to authenticate using kerberos automatically when the Jenkins server replies “401 Unauthorized” and indicates it supports kerberos. That is, kerberos authentication should work automagically. For a quick test, just try the following.

```
import jenkins

server = jenkins.Jenkins('http://localhost:8080')
print server.jobs_count()
```

Note

Jenkins as such does not support kerberos, it needs to be supported by the Servlet container or a reverse proxy sitting in front of Jenkins.

Example 3: Working with Jenkins Jobs

This is an example showing how to create, configure and delete Jenkins jobs.

```
server.create_job('empty', jenkins.EMPTY_CONFIG_XML)
jobs = server.get_jobs()
print jobs
my_job = server.get_job_config('cool-job')
print(my_job) # prints XML configuration
server.build_job('empty')
server.disable_job('empty')
server.copy_job('empty', 'empty_copy')
server.enable_job('empty_copy')
server.reconfig_job('empty_copy', jenkins.RECONFIG_XML)

server.delete_job('empty')
server.delete_job('empty_copy')

# build a parameterized job
# requires creating and configuring the api-test job to accept 'param1' & 'param2'
server.build_job('api-test', {'param1': 'test value 1', 'param2': 'test value 2'})
last_build_number = server.get_job_info('api-test')['lastCompletedBuild']['number']
build_info = server.get_build_info('api-test', last_build_number)
print build_info

# get all jobs from the specific view
jobs = server.get_jobs(view_name='View Name')
print jobs
```

Example 4: Working with Jenkins Views

This is an example showing how to create, configure and delete Jenkins views.

```
server.create_view('EMPTY', jenkins.EMPTY_VIEW_CONFIG_XML)
view_config = server.get_view_config('EMPTY')
views = server.get_views()
server.delete_view('EMPTY')
print views
```

Example 5: Working with Jenkins Plugins

This is an example showing how to retrieve Jenkins plugins information.

```
plugins = server.get_plugins_info()
print plugins
```

The above example will print a dictionary containing all the plugins that are installed on the Jenkins server. An example of what you can expect from the `get_plugins_info()` method is documented in the API reference doc.

Example 6: Working with Jenkins Nodes

This is an example showing how to add, configure, enable and delete Jenkins nodes.

```
server.create_node('slave1')
nodes = get_nodes()
print nodes
node_config = server.get_node_info('slave1')
print node_config
server.disable_node('slave1')
server.enable_node('slave1')

# create node with parameters
params = {
    'port': '22',
    'username': 'juser',
    'credentialsId': '10f3a3c8-be35-327e-b60b-a3e5edb0e45f',
    'host': 'my.jenkins.slave1'
}
server.create_node(
    'slave1',
    nodeDescription='my test slave',
    remoteFS='/home/juser',
    labels='precise',
    exclusive=True,
    launcher=jenkins.LAUNCHER_SSH,
    launcher_params=params)
```

Example 7: Working with Jenkins Build Queue

This is an example showing how to retrieve information on the Jenkins queue.

```
server.build_job('foo')
queue_info = server.get_queue_info()
id = queue_info[0].get('id')
server.cancel_queue(id)
```

Example 8: Working with Jenkins Cloudbees Folders

Requires the Cloudbees Folders Plugin for Jenkins.

This is an example showing how to create, configure and delete Jenkins folders.

```
server.create_job('folder', jenkins.EMPTY_FOLDER_XML)
server.create_job('folder/empty', jenkins.EMPTY_FOLDER_XML)
server.copy_job('folder/empty', 'folder/empty_copy')
server.delete_job('folder/empty_copy')
server.delete_job('folder')
```

Example 9: Updating Next Build Number

Requires the Next Build Number Plugin for Jenkins.

This is an example showing how to update the next build number for a Jenkins job.

```
next_bn = server.get_job_info('job_name')['nextBuildNumber']
server.set_next_build_number('job_name', next_bn + 50)
```

Example 9: Working with Build Promotions

Requires the Promoted Builds Plugin for Jenkins.

This is an example showing how to create, configure and delete a promotion process for an existing job.

The job in this example is named *prom_job* and it needs to have this config xml snippet before creating the promotion:

```
<properties>
   <hudson.plugins.promoted__builds.JobPropertyImpl>
   <activeProcessNames>
      <string>prom_name</string>
    </activeProcessNames>
  </hudson.plugins.promoted__builds.JobPropertyImpl>
</properties>
```

where *prom_name* is the name of the promotion that will get added to the job.

```
server.create_promotion('prom_name', 'prom_job', jenkins.EMPTY_PROMO_CONFIG_XML)
server.promotion_exists('prom_name', 'prom_job')
print server.get_promotions('prom_job')

server.reconfig_promotion('prom_name', 'prom_job', jenkins.PROMO_RECONFIG_XML)
print server.get_promotion_config('prom_name', 'prom_job')

server.delete_promotion('prom_name', 'prom_job')
```

Example 10: Waiting for Jenkins to be ready

It is possible to ask the API to wait for Jenkins to be ready with a given timeout. This can be used to aid launching of Jenkins and then waiting for the REST API to be responsive before continuing with subsequent configuration.

```
# timeout here is the socket connection timeout, for each connection
# attempt it will wait at most 5 seconds before assuming there is
# nothing listening. Useful where firewalls may black hole connections.
server = jenkins.Jenkins('http://localhost:8080', timeout=5)

# wait for at least 30 seconds for Jenkins to be ready
if server.wait_for_normal_op(30):
    # actions once running
    ...
else:
    print("Jenkins failed to be ready in sufficient time")
    exit 2
```

Note that the timeout arg to *jenkins.Jenkins()* is the socket connection timeout. If you set this to be more than the timeout value passed to *wait_for_normal_op()*, then in cases where the underlying connection is not rejected (firewall black-hole, or slow connection) then *wait_for_normal_op()* may wait at least the connection timeout, or a multiple of it where multiple connection attempts are made. A connection timeout of 5 seconds and a wait timeout of 8 will result in potentially waiting 10 seconds if both connections attempts do not get responses.