GoCD Python API client Documentation

Release 1.1.0

Grigory Chernyshev

Apr 13, 2017

Contents

CHAPTER 1

Intro

This library is a high level python API wrapper upon ThoughtWorks GoCD REST API.

Here are some highlights:

- it's created and designed with simplicity in mind, so it would be easy to start using it.
- there is only one class, that you would need to initialize and work with.
- pipelines are linked together, so you can iterate over predecessors or descendants of a given pipeline.
- it's very close to the original REST API implementation from ThoughtWorks.
- every class and function is annotated with `@since` decorator, which gives possibility to check at runtime whether specific feature is supported on given server version.
- it's possible to use latest library version with any GoCD server version, even if some parameters or headers are different: we have special methods to pass correct parameters depending on the server version.
- *every* version of GoCD is meticulously tested, thanks to releases of it in Docker container. Here is a list of versions supported so far:
  - 16.1.0
  - 16.2.1
  - 16.3.0
  - 16.6.0
  - 16.7.0
  - 16.8.0
  - 16.9.0

  Older version should work as well, but as they are not supported and there is no Docker images for them, you should use them on your own risk.

CHAPTER 2

Contents:

Yet another GoCD python client

Introduction

This library is a high level python API wrapper upon ThoughtWorks GoCD REST API. From the official documentation:

> Go Continuous Delivery is continuous integration/deployment server, which helps you automate and streamline the build-test-release cycle for worry-free.

Using it, you can access to internals of the Pipelines, check their statuses, download Artifacts and more. It is designed with maximum comfort and productivity from the user perspective, so it should be very easy to use it in your own project. More information is available at official documentation

Installation

```
$ pip install yagocd
```

Quick example

```
from yagocd import Yagocd

go = Yagocd(
    server='https://example.com',
    auth=('user', 'password'),
    options={
        'verify': False  # skip verification for SSL certificates.
```

```
    }
)

for pipeline in go.pipelines:    # Iterate over all pipelines
    for instance in pipeline:    # Iterate over each instance of some pipeline
        for stage in instance:   # Iterate over each stages of some pipeline instance
            for job in stage:    # Iterate over each job of some pipeline
                for root, folder, files in job.artifacts:  # Iterate over artifacts␣
↪like `os.walk` manner
                    for artifact in files:
                        print(artifact.data.url)

                # print property of each job
                for key, value in job.properties.items():
                    print("{} => {}".format(key, value))
```

Different implementations of GoCD API

Here is list of similar projects, that implements GoCD API:

- py-gocd [python]: A Python API for interacting with Go Continuous Delivery
- gocdapi [python]: A Python API for accessing resources and configuring Go (thoughtworks) continuous-delivery servers
- gomatic [python]: A Python API for configuring GoCD
- goapi [ruby]: Go (http://www.go.cd) API ruby client
- gocd-api [NodeJS]: Access http://www.go.cd API via nodeJS
- gocd-api [GoLang]: An implementation of the gocd API

Licence

MIT

Usage

Here you can find quick introduction into the usage of different components of the library. It's far from complete and is not intended to cover all functionality. To get more information you can address modules documentation.

Initialization

To use GoCD Python API client in a project you have to import it first:

```
from yagocd import Yagocd
```

Next, you can create an instance of the client for connecting to the server:

```
client = Yagocd(
      server='http://localhost:8153/',
      auth=('admin', 'secret')
  )
```

Here, we created a client, which would connect to the server running on a localhost, port number *8153*. Please note, that you don't have to put `go/` context path to the `server` variable. `auth` is a tuple of login and password for connecting to the server. There is possibility to set additional parameters during initialization, passing them to the `options` variable:

- `context_path`: server context path to use (default is `go/`).
- `verify`: verify SSL certs. Defaults to `True`.

Managers

`Yagocd` is the only class that you usually would need to work with, therefore it gives access to all other resources. Each resource have a manager associated with it. Managers represents a classes with list of public methods that could be used by client code. In some cases those methods return some objects that could inspected or used. Majority of those objects are inherited from `Base` class, which has special `data` attribute for accessing to object's data via dot notation, e.g.:

```
pipeline = client.pipelines['Shared_Services']
print(pipeline.data.name)
>> Shared_Services
```

Further you would find examples of using some of those managers.

Pipelines

Pipelines are one of the core components in GoCD architecture. The pipelines API allows users to view pipeline information and operate on it.

Listing pipelines

Here how you can get list of them:

```
pipelines = client.pipelines.list()
```

Or you can use iteration instead `list()`:

```
for pipeline in client.pipelines:
  print(pipeline)
```

Beware though, listing all pipelines could be heavy operation in case you have zillions of pipelines of your server.

Getting specific pipeline

If you know the name of the pipeline in advance, you can use `get()` or array-like access syntax:

```
pipeline = client.pipelines.get('Shared_Services')
# OR
pipeline = client.pipelines['Shared_Services']
```

As there no separate method for getting specific pipeline, current implementation of `get()` is based on filtering the results of the `list()`.

Pipelines are linked together

There is an interesting feature when you are working with pipelines through Yagocd library: as pipelines could have relations between each other, this information is used to build a graph of dependencies between them. You can use this in your code like this:

```
pipeline = client.pipelines.get('Consumer_Website')
for child in pipeline.predecessors:
  print(child)

for parent in pipeline.descendants:
  print(parent)
```

`predecessors` and `descendants` are properties for accessing appropriate relations. By default only direct relations are fetched. If you need to get all of them, you can use `get_predecessors(transitive=True)` and `get_descendants(transitive=True)` methods correspondingly.

Getting instance of a pipeline

First of all, there is a difference between pipeline and pipeline instance: first is a descriptor or configuration of a pipeline. You can schedule execution of it or get it's history. Pipeline instance is an execution of a given pipeline. You can check it's logs, for example.

History would give you execution history of a given pipline. To get pipeline history, i.e. pipeline instances, you can use `history()` or `full_history()`. Latter would not stop after first 10 items, but would iterate over all executions of a given pipeline.

It's possible to use `last()` method, which would return you the most recent pipeline instance.

Finally, it's possible to get instance of a pipeline by it's counter using `get()` method and passing counter as a parameter.

Accessing stages of a pipeline instance

As pipeline could have one or more stages, you might want to access this information. You can use `stages()` method to get list of available stages:

```
pipeline = client.pipelines.get('Consumer_Website')
pipeline_instance = pipeline.last()
stages = pipeline_instance.stages()
# OR
for stage in pipeline_instance:
  print(stage)
```

If you are interested in specific stage, you can get it by name:

```
stage = pipeline_instance['stage_name']
```

Stages

The stages API allows users to view stage information and operate on it.

Accessing jobs of a stage instance

Stage instance gives you access to it's job instances:

```
stage_instance = client.stages.get(
  pipeline_name='Consumer_Website',
  pipeline_counter=31,
  stage_name='Commit',
  stage_counter=1
)

jobs = stage_instance.jobs()
# OR
for job in stage_instance:
  print(job)
```

If you are interested in specific job, you can get it by name:

```
job = stage_instance['job_name']
```

Jobs

The jobs API allows users to view job information.

Accessing artifacts

You can list available artifacts for specific job:

```
artifacts = job.artifacts.list()
# OR
for artifact in job.artifacts:
  print(artifact)
```

Each artifact could have some files or directories in it. You can iterate over them and get it's content:

```
for filename in artifact.files():
  content = filename.fetch()
```

If you know the name of the file or the directory, you can download it like this:

```
file_content = job.artifacts['/path/to/filename.txt']
dir_zip_content = job.artifacts['/path/to/folder.zip']
```

Accessing properties

Job could have properties set during build. The are represented in dictionary-based form. You can iterate over them like this:

```
for name, value in job.properties.items():
  print(name, value)
```

Or you can read value of specific property by it's name:

```
value = job.properties['property_name']
```

Release History

1.0.0 (2017-04-13)

Improvements - Added elastic profile support. - Added package repositories API support. - Added package management API support. - Added encryption API support. - Improvements in *ValueStreamMap*. - Added enumerations (constants) for stage result and state. - Reworked artifact manager (breaking change!) - ETag is now part of resource object (breaking change!)

Miscellaneous - Added support for 16.11.0 version. - Added support for 17.01.0 version. - Added support for 17.02.0 version. - Added support for 17.03.0 version. - Refactored managers urls and parameters.

0.4.4 (2016-12-01)

Improvements - Client's manager are now initialized just once, which makes possible to cache the results of their calls.

0.4.3 (2016-11-23)

Improvements - Improvements in *ValueStreamMap*. - Added possibility to programmatically disable version check in *Since* decorator, using *ENABLED* flag.

Miscellaneous - Added support for 16.10.1 version.

0.4.2 (2016-10-28)

Improvements - ValueStreamMap: put dictionary instead of StageInstance.

0.4.1 (2016-10-09)

Improvements - Added custom exception error, which outputs error in clear format. - Added support for pluggable SCM materials API. - Added support for template API. - Improvements in *ValueStreamMap*.

Miscellaneous - Documentation updated. - Docker image updated, which used in testing. - Added support for 16.10.0 version.

0.4.0 (2016-10-01)

Improvements

- Added support for pipeline config API.
- Added support for version API.
- Added support for plugin info API.
- Added support for environments API.
- Added methods for getting different internal information (undocumented): *support* and *process_list*.

- Added magic methods for iterating and key based access for some classes.
- All classes and their methods are now decorated with *@since* decorator, which adds possibility to check at run-time whether given functionality already supported in the GoCD server and let's dynamically select correct headers.

Testing

- Now tests are executed for GoCD version, running in Docker container, which add possibility to test for any available version of the server. Also cassettes are also saved individually for each GoCD version.
- Added testing for PEP8 and other checks via *flake8*.

0.3.2 (2016-07-26)

Improvements

- Added support of *value_stream_map* functionality.

Bugfixes

- Fix return value of *Artifact.fetch* method from text to binary.

0.2.0 (2016-05-24)

Improvements

- Added support of getting server version through parsing */about* page.
- Added *Confirm: true* header to some API calls.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search